(2)

AD-A187 210

# DIGITAL CONTROL OF THE CZOCHRALSKI GROWTH OF GALLIUM ARSENIDE - CONTROLLER SOFTWARE REFERENCE MANUAL

Arizona State University
Semiconductor Materials Research Laboratory
Tempe, AZ 85287

AFOSR-TR- 87-1542

July 15, 1987

Scientific Report, October 1, 1985 - March 31, 1987

ARPA Order Nos.: 5187
Contract Nos.: F49620-85-C-0010
Contract Effective Dates: 10/1/84
Contract Expiration Dates: 3/31/86

Program Manager: G. H. Schwuttke
(602) 965-2672
Contract Monitor: Gary Witt, AFOSR
(202) 767-4931

[redacted]

Prepared for
Defense Advanced Research Projects Agency
1400 Wilson Blvd.
Arlington, VA 22209

Air Force Office of Scientific Research
AFOSR/NE
Bolling AFB, DC 20332

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

ADA187210

REPORT DOCUMENTATION PAGE

| | |
|---|---|
| 1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED | 1b. RESTRICTIVE MARKINGS |
| 2a. SECURITY CLASSIFICATION AUTHORITY | 3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release, distribution unlimited |
| 2b. DECLASSIFICATION/DOWNGRADING SCHEDULE | |
| 4. PERFORMING ORGANIZATION REPORT NUMBER(S) | 5. MONITORING ORGANIZATION REPORT NUMBER(S) AFOSR-TR- 87-1542 |
| 6a. NAME OF PERFORMING ORGANIZATION Arizona State University Semiconductor Materials Lab. — 6b. OFFICE SYMBOL (If applicable) SPA | 7a. NAME OF MONITORING ORGANIZATION Air Force Office of Scientific Research |
| 6c. ADDRESS (City, State and ZIP Code) Arizona State University Tempe, AZ 85287 | 7b. ADDRESS (City, State and ZIP Code) AFOSR/NE Bolling AFB, D.C. 20332 |
| 8a. NAME OF FUNDING/SPONSORING ORGANIZATION DARPA — 8b. OFFICE SYMBOL (If applicable) ~~DSO~~ NE | 9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER F49620-86-C-0010 |
| 8c. ADDRESS (City, State and ZIP Code) DARPA/DSO 1400 Wilson Blvd Arlington, VA 22209 | 10. SOURCE OF FUNDING NOS. PROGRAM ELEMENT NO. 63320C; PROJECT NO. DARPA; TASK NO. 9099; WORK UNIT NO. |
| 11. TITLE (Include Security Classification) Digital Control of the Czochralski Growth of Gallium Arsenide - Controller Software Reference Manual | |
| 12. PERSONAL AUTHOR(S) Karl Riedling | |
| 13a. TYPE OF REPORT Scientific Final — 13b. TIME COVERED FROM 10/1/85 TO 3/31/87 | 14. DATE OF REPORT (Yr., Mo., Day) 87-7-15 — 15. PAGE COUNT 182 |
| 16. SUPPLEMENTARY NOTATION | |

| 17. COSATI CODES | | | 18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number) |
|---|---|---|---|
| FIELD | GROUP | SUB. GR. | Digital Control |
| | | | GaAs |
| | | | Reference Manual |

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

This volume provides a complete description of the structure and the operation of the specific controller software developed for ASU's digital Czochralski Growth Control System (CGCS) for compound semiconductors. The manual is primarily intended for use by advanced programmers and crystal growth specialists. In four main chapters, the Controller Software Reference Manual discusses the design considerations applied to digital LEC crystal growth control, gives a short overview over the growth controller computer hardware and operating system environment, describes the functions of the CGCS from an operator's point of view, and delineates the internal operations of the controller software by discussing the controller software and algorithms. Various appendices provide tables of controller software tasks, routines, and variables, file format information, and lists of system messages and error codes.

| | |
|---|---|
| 20. DISTRIBUTION/AVAILABILITY OF ABSTRACT UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS ☐ | 21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED |
| 22a. NAME OF RESPONSIBLE INDIVIDUAL [illegible] | 22b. TELEPHONE NUMBER (Include Area Code) (202) 767-4931 — 22c. OFFICE SYMBOL NE |

DD FORM 1473, 83 APR EDITION OF 1 JAN 73 IS OBSOLETE

Scientific Report

# DIGITAL CONTROL OF THE CZOCHRALSKI GROWTH OF GALLIUM ARSENIDE - CONTROLLER SOFTWARE REFERENCE MANUAL

Sponsored by

Defense Advanced Research Projects Agency

G. H. Schwuttke
Principal Investigator
(602) 965-2672

Arizona State University
Semiconductor Materials Laboratory
College of Engineering & Applied Sciences
Tempe, Arizona 85287

## ABSTRACT

This volume provides a complete description of the structure and the operation of the specific controller software developed for ASU's digital Czochralski Growth Control System (CGCS) for compound semiconductors. The manual is primarily intended for use by advanced programmers and crystal growth specialists. In four main chapters, the Controller Software Reference Manual discusses the design considerations applied to digital LEC crystal growth control, gives a short overview over the growth controller computer hardware and operating system environment, describes the functions of the CGCS from an operator's point of view, and delineates the internal operations of the controller software by discussing the controller software and algorithms. Various appendices provide tables of controller software tasks, routines, and variables, file format information, and lists of system messages and error codes. Keywords: Gallium arsenides,

TABLE OF CONTENTS

## EXECUTIVE SUMMARY

A digital controller system for the Czochralski growth of gallium arsenide single crystals is presented. Digital growth control was chosen because of its essential advantages over the standard analog approach:

* Better reproducibility of process parameters and control actions.

* A higher degree of flexibility with regard to operation procedures and process parameters.

* Powerful process automation.

* Expanded process data logging facilities.

The digital Czochralski Growth Control System (CGCS) is based on a microcomputer built around an Intel 8085 microprocessor. The system hardware consists of commercial OEM components; the microcomputer features 16 KBytes of Read Only Memory (ROM) and 56 KBytes of Random Access Memory (RAM), an Intel 8231 Numeric Processor, two industrial standard 8" single sided, single density flexible disk drives, and the Analog/Digital and Digital/Analog Converters and Input/Output (I/O) hardware which it requires to interface to the Czochralski puller. In addition to a console CRT terminal, a line printer is provided. The controller computer was designed as a multi-purpose unit which permits, in addition to the actual process control, to execute auxiliary programs for the maintenance of disks and disk files, and for the preparation and evaluation of growth runs. The operating system used is Intel's Real-Time Multitasking Executive iRMX-80; a special system environment, RXISIS-II, was developed for the execution of utility and support programs.

The CGCS is wired to monitor process data in parallel to the standard analog growth controller; its output can alternatively replace the analog controller's output. For reasons of simplicity, it uses part of the analog system's signal conditioning and output circuitry. In particular, it provides the analog motor speed and the heater power controllers with speed and power setpoints. The digital system can be operated in the following modes each of which is a superset of the preceding one:

(1) Monitoring: The CGCS collects data from the puller which can be displayed and recorded, but it does not control the puller.

(2) Manual: The CGCS controls the growth process but allows only to enter setpoints for the primary process parameters

(temperatures, motor speeds). No closed-loop diameter control is possible.

(3) Diameter: This mode includes closed-loop diameter control, based on the standard weighing method. Special algorithms compensate for the buoyancy effects caused by the encapsulation melt.

(4) Diameter/ASC: In addition to the above features, an anomaly compensation technique is used, which makes the diameter calculated by the CGCS more reliable.

(5) Automatic: A special algorithm permits to maintain the crystal-melt interface at a constant location within the heater, regardless of the amount of melt depleted due to crystal growth.

The CGCS software allows to modify any parameter, including the parameters of controller loops, by direct operator commands. Parameters may be "ramped" within an arbitrary time from their current to their intended final values. Commands may be recorded on special disk files which may be edited and replayed as Macro commands during a later run; the sequence and timing of the recorded commands is exactly reproduced. These recorded commands can, still, be arbitrarily interspersed with new commands entered on the console; the resulting command sequence may be recorded again, which gives the system a learning ability. Macro command files may comprise any number of commands and can easily be invoked by name. A special feature permits to execute Macro commands conditionally, i.e., if and when a specified relation between an arbitrary system parameter and a constant value is reached. These features allow to execute certain stages of the crystal growth process automatically, without the necessity of operator interactions. Although it is not yet possible to automate the entire growth process because the process data available to the CGCS is not sufficient, we have obtained a considerable improvement over the level of automation of the standard analog controller.

Great emphasis was put on the design of the operator-machine interface: A specially formatted CRT console screen provides information about all data measured by the CGCS. Command entry is interactive, with as much flexibility as possible with regard to the format of the commands. Several help menus and extensive command prompts guide the operator. The dialog between the operator and the CGCS can be recorded either on disk, or on a line printer; each item is tagged with the time when it was issued. This permits, in conjunction with the data recording facilities of the CGCS, to trace the effects of a particular operation; the data taken during a run can be used for various process analysis and modelling approaches.

## HOW TO USE THIS DOCUMENTATION

This documentation details the internal operation of the CGCS software. It refers to hardware functions where necessary but should <u>not</u> be considered a hardware manual. Although it does <u>not</u> primarily give operating instructions, it may contribute to the user's understanding of the system's operations, and it may therefore clarify some points in question. Readers proficient in Fortran may find it advantageous to have the program listings at hand (which are very extensively commented, too); frequently, references are made within this manual to the names of program variables or routines. It is, however, not necessary to study the source programs to understand this documentation. A number of additional documentations are listed in Appendix A which provide more detailed information about items which could only be mentioned here.

## CGCS PROGRAM VERSIONS

This issue of the Software Reference Manual is based upon version 2.3 of the Czochralski Growth Control System. This section describes the "evolution" of the program by listing the features newly introduced with each release.

Version 1.3: (October 19, 1985)

(Version 1.3 was the first program release actually used for growing gallium arsenide crystals.)

Version 1.4: (December 5, 1985)

(1) INITIALIZE sets the diameter setpoint to the seed diameter. (This feature was discontinued from Version 2.1 on.)

(2) The Diameter evaluation routines check for zero seed lift speed and disable diameter calculation in this case.

(3) An automatic RESET is executed when required.

(4) The calculated Diameter is recorded in the Data file.

Version 1.5: (February 1986)

(1) RESET permits the entry of a reset value for the Crystal Weight and/or the Length Grown. (The effect of RESET on the Crystal Weight is a new feature of this release.)

(2) The length of the crystal stored by the buoyancy compensation part of the diameter calculation routine was increased from 37.5 millimeters to 75 millimeters. The thickness of one "slice" is approximately 0.5 mm; the maximum permitted seed travel speed exceeds than 200 mm/h.

(3) The actual Diameter value is automatically copied to the Diameter setpoints when any Diameter controlled mode is entered.

Version 1.6: (February 18, 1986)

(1) The Data Dump facility was newly introduced. Extra records are written to the Data file in case of an error detected by the Diameter Evaluation routines.

(2) The crystal diameter is evaluated with the actual growth rate rather than with the (actual) seed lift speed.

(3) The Diameter Evaluation routines are able to recover from Speed Overflow errors automatically. (In previous versions, such errors disabled the diameter evaluation permanently; a RESET command was required to recover from this condition.)

Version 2.0: (April 11, 1986)

(1) The number of ramping channels was increased from 8 to 20.

(2) The maximum number of Conditional commands is 8 rather than 2. Conditional commands entered while already 8 commands are pending are ignored. (In earlier versions, a Conditional Macro command issued while already two Conditional commands were pending replaced the older one).

(3) A Selective CLEAR command was introduced which permits to remove only those Conditional Macro commands from the Conditional Command queue which pertain to a specified Variable.

(4) The PLOT feature was implemented, providing 8 analog channels for the output of arbitrary INTEGER*2 parameters, plus a set of pre-processed system parameters (Temperatures, Diameter error, Growth Rate, and Crucible Position error).

(5) 8 INTEGER*2 DUMMY locations were provided as a Macro command scratchpad.

(6) The CGCS can be put into a TEST mode. (Program patches (in ANACNT) were required in previous versions to execute run simulations.)

Version 2.1: (October 13, 1986)

(1) An algorithmic error in the Diameter Evaluation routine was corrected which resulted in a relative error of the calculated Growth Rate in the order of 10 percent.

(2) The buoyancy compensation routines were re-designed. In particular, a new interpolation algorithm is used for the determination of the crystal diameter at the top surface of the boric oxide encapsulant. A partial compensation of the effects caused by melt recession at the end of the growth process was provided.

(3) Two new operation modes of the PID controller routine are available with release 2.1. They provide different approaches for a safe "anti-windup" function which improves

the dynamic behavior of the controller in its output limited regime.

(4) The scaling of the Heater and Base Temperature output to the chart recorder was improved. A Variable-defined output range permits a flexible adaptation of the chart recorder output to various operating conditions.

(5) A timeout for the printer interface was activated. This feature prevents a defective or unselected printer from suspending the operation of the system.

Version 2.2: (October 24, 1986)

(1) A new, more stable diameter interpolation algorithm replaces part of the procedures introduced with Version 2.1.

(2) The melt recession compensation algorithms were improved. A numeric parameter permits to adapt the Diameter Evaluation routines to arbitrary degrees of melt recession.

(3) The (square of the) crystal diameter stored in a table internal to the Diameter Evaluation routine is checked for excessive deviations with respect to its previous value, and adjusted accordingly.

(4) A check for a possible boric oxide encapsulant height overflow permits to run the CGCS safely with increased boric oxide charges.

(5) Conditional command checking is disabled for several seconds after a new (Conditional or unconditional) Macro command was started, in order to make sure that at least the first command of a Macro file can be executed in any case.

(6) An improved Macro command execution guarantees the proper execution of Macro commands even in the case of transient disk errors.

(7) The generation of the Data disk file which was performed by two tasks in previous versions (one, collecting data, and one, writing it to disk) was concentrated in one single task. This measure provides the memory space required for the installation of the other software enhancements and reduces the probability of a temporary system deadlock due to a lack of pool memory, with the penalty of a possible minor record timing inaccuracy in the case of very short intervals between Data file records.

<u>Version 2.3:</u> (December 5, 1986)

(1) A periodic memory check was provided in this release, comprising the RAM resident main program code.

# 1. DIGITAL CONTROL OF CZOCHRALSKI GaAs CRYSTAL GROWTH

## 1.1 INTRODUCTION

The Czochralski process is gaining increased importance not only for the growth of high purity silicon crystals but also for the large scale production of compound semiconductors like gallium arsenide. Although Czochralski grown GaAs crystals do not yet reach low dislocation densities comparable to those obtainable with the major competitor process, the Bridgeman technique, the Czochralski process offers, nevertheless, significant advantages over boat growth processes:

* The stoichiometry and the purity of Czochralski-grown crystals is superior to the properties of boat-grown ones. Semi-insulating substrates can be obtained with less or even without chromium doping.

* The Czochralski process is better suited for a large scale production, and it is therefore cheaper.

A Czochralski puller (Fig. 1) consists essentially of a heated crucible made of quartz or boron nitride which contains the semiconductor melt. A small single crystal rod, the seed, is immersed into the melt and slowly lifted. The melt whose temperature is kept slightly above the semiconductor's melting point solidifies at the interface to the seed; with the proper temperature distribution and seed lift speed, a cylindrical single crystal can be grown whose crystallographic orientation is determined by the orientation of the seed. The crucible and the seed are rotated in opposite directions in order to minimize the influence of potential inhomogeneities of the temperature distribution inside the furnace. An inert atmosphere, usually argon, prevents the oxidation of the melt and of the crystal.

An additional problem is caused by the fact that compound semiconductors like GaAs tend to dissociate at higher temperatures. The two components are bound together only loosely, and the one with the higher gas pressure (in our case, arsenic) is evaporated to a greater degree than the other (gallium), which results in intolerable deviations from stoichiometry and, in consequence, in bad electrical characteristics. While the miscellaneous variations of the Bridgeman process employ hermetically sealed ampoules made of quartz to prevent the loss of the volatile component, two approaches are used with the Czochralski process, either individually or combined: First, the pressure of the inert atmosphere inside the puller is increased to several hundred psis in order to counterbalance the arsenic vapor pressure, and, second, the semiconductor melt and the part of the crystal next to it are encapsulated in a vitreous melt of boric oxide.

Technical applications of semiconductor single crystals require a defined, and preferably cylindrical, shape of the crystal ingots which have to be sliced into wafers with given dimensions. Semiconductor crystal growth implies, therefore, an efficient control of the diameter of the crystals grown. Neither must the diameter drop below a minimum value (which would prohibit cutting a wafer with the specified diameter), nor should the diameter exceed its nominal value too much since the excess material is wasted as it must be ground away before the ingot is sliced into wafers. Conventional Czochralski pullers for compound semiconductors determine the diameter of the growing crystal from the increase of its weight per unit time which is obviously proportional to the crystal volume solidified per time. Taking a constant pull rate, i.e., a constant height of the incremental solid cylinder, for granted, this volume is proportional to the square of the crystal diameter. Diameter control can be effected by changing the temperature of the melt and/or the pull rate appropriately: The solidification of the molten semiconductor material generates heat which must be removed from the interface between the crystal and the melt in order to permit a continuous growth. The amount of heat which can be removed from the interface is, however, determined by the geometry of the furnace and of the crystal, and it is more or less constant. Increasing the temperature of the melt permits therefore less material to solidify, which results in a reduction of the crystal diameter if the pull rate is kept constant. On the other hand, an increase of the pull rate while the melt temperature is maintained has the consequence that the roughly constant volume of semiconductor material which can be solidified per unit time has to be stretched out to a longer and narrower cylinder, thus reducing the crystal diameter, and vice versa. (Compound semiconductors are, however, generally grown with temperature based diameter control since changes of the pull rate tend to deteriorate the material quality.)

A basic compound semiconductor puller features, therefore, the following elements (compare Fig. 1):

(1) A temperature controlled heater.

(2) Four speed controlled motors which are in charge of

(a) the rotation of the crucible;

(b) the rotation of the crystal;

(c) the seed lifting motion; and

(d) the lifting of the crucible which keeps the interface between the melt and the solid crystal at the same location within the heater in order to guarantee a constant temperature profile at the critical interface region.

(3) An electronic balance which permits to determine the crystal's weight and the weight increment; the latter signal can be used to control the heater temperature in order to maintain a defined crystal diameter.

Conventional compound semiconductor Czochralski pullers use analog electronic circuits to control the heater temperature and the motor speeds. Although this is an obvious approach (since all input and output parameters are inherently analog signals), there are several severe drawbacks associated with analog control circuitry:

(1) Analog controllers usually obtain their control parameters (e.g., the gain of a controller amplifier) from the setting of a potentiometer. It is not only difficult (and, frequently, impossible) to modify such parameters dynamically during a growth run although this might be desirable, it is also problematic to return to exactly the same settings which were used during earlier experiments once a parameter was changed.

(2) Despite of the fact that there are analog controllers on the market which feature a high degree of automation, the actual growth process is basically determined by the human operator. The high degree of human interaction, combined with the questionable repeatability of an analog system, makes it difficult to provide exactly reproducible growth conditions for different growth runs.

(3) Crystal growth is, in fact, a very complex and not yet sufficiently understood process. A better understanding of the process which is the prerequisite for any process improvement can, however, be only based upon the thorough analysis of actual growth data. The logging of process data, particularly, of a greater number of data channels, is a very awkward procedure in an analog system; usually, crystal growers have to be content with three or so data channels logged on an analog chart recorder.

All these considerations favor the introduction of digital computer control for Czochralski crystal growth. A numerically based control permits not only absolute reproducibility of process parameters; it can much more readil, be interfaced with automation approaches, and it permits, last but not least, the recording of growth data in a form suitable for later computer analysis.

## 1.2 BASIC CONCEPT OF A DIGITAL CONTROLLER FOR GaAs CZOCHRALSKI GROWTH

The basic target of the current project towards digital control of the Czochralski process for GaAs crystal growth was to replace the standard analog controller supplied by Cambridge Instruments, the company that built and delivered the puller proper, by a suitable computer-based controller. Since the complete setup is basically an experimental one, great emphasis had to be put on versatility and flexibility. Therefore, the approach shown in Fig. 2 was chosen:

The digital controller is connected in parallel to the standard analog one. Both systems monitor in parallel the output signals provided by the puller's sensors. Switches (actually, relays driven by the digital controller) permit to apply control signals to the puller either from the analog or from the digital controller. This allows, in conjunction with the proper software support, to switch control between both systems even during a growth run, which is particularly important during the setup and tuning of the digital controller. For reasons of simplicity, the digital system uses part of the signal conditioning circuitry and the motor controller and heater SCR circuits of the standard analog console. The digital system supplies, therefore, only motor speed and heater power setpoints; the standard analog controller's circuitry provides closed-loop motor speed and heater power control. Furthermore, only those functions of the puller which directly affect the growth conditions are digitally controlled. Although the digital system is therefore not capable of running the puller without the standard analog circuitry, this restriction to the most important operations permits to concentrate on features which are essential for the crystal growth, and facilitates the hardware and software implementation of the digital controller.

The following analog signal sources were chosen to be monitored by the digital controller, in parallel to the analog Cambridge console:

(1) Three thermocouples, measuring up to three heater zone temperatures. (Currently, only a single-zone heater is in use.)

(2) Four tachometers which are connected to the four motors for seed and crucible lift and rotation. (In contrast to the Cambridge Instruments terminology of "crystal" lift and rotation, we are using "seed" lift and rotation within this documentation and within the software, in order to avoid confusions of "crystal" and "crucible", particularly in abbreviations.)

(3) Up to three wattmeters which are connected to the puller's heater(s).

(4) The weight gauge monitoring the crystal weight.

(5) An analog differentiator circuit which generates a signal proportional to the first derivative of the crystal weight with regard to time. Determining the differential weight with an analog circuit rather than calculating it numerically from the plain weight was found advantageous because the crystal weight changes very slowly due to the slow growth of compound semiconductors. In order to allow to calculate the differential weight from the plain weight in practical time intervals with a reasonable resolution, the weight signal would have, therefore, required an extremely high analog-to-digital resolution, in excess of 20 bits. Suitable hardware is hardly commercially available, at least, for a reasonable price. In contrast, a 14 bit resolution is sufficient for all signals, including the plain weight, if analog weight differentiation is used.

(6) Two potentiometers which return voltages proportional to the current positions of the seed and the crucible, respectively.

(7) A thermocouple measuring the temperature at the bottom of the crucible ("base temperature").

(8) A pressure gauge sensing the pressure inside the puller's vessel.

(9) The "contact device" which is basically an ohmmeter circuit which monitors the resistance between the seed and the melt. This resistance drops from infinity to a certain value when the seed touches the (semiconducting) boric oxide encapsulation melt, and it drops further when contact between the seed and the actual semiconductor melt is established.

(10) Eight spare channels which can be used to record additional information (for example, the outputs of auxiliary thermocouples) together with growth data.

## 1. Digital Control of Czochralski GaAs Crystal Growth

The signals which are supplied by the digital controller as replacements for the analog system's outputs are:

(1) Three heater SCR control voltages, anticipating a three-zone heater. (Currently, only one control voltage is used.)

(2) Four speed control voltages for the seed and crucible lift and rotation motors.

(3) Up to eight internal parameters can be submitted to a digital/analog conversion; the resulting eight analog signals can be recorded on a suitable multi-channel chart recorder.

In addition, digital signals are monitored by the digital controller and provided for the puller:

(1) Four motor direction signals: They are required, in addition to the (unipolar) speed control voltages, in order to determine the direction of motor motion (up or down, or clockwise or counterclockwise). The same control signals are also used within the standard analog controller; these signals generated by the analog circuitry are monitored by the digital system to provide complete status information.

(2) One master control signal: All control signal changeover relays are energized to select the digital system as a control signal source if this signal is present. Otherwise, the analog controller is in full charge of the puller. This is obviously an output-only signal of the computer system.

The quasi-parallel operation of the analog and the digital controllers suggests a multi-step approach for the implementation of the computer-based system which is, indeed, supported by the digital controller software. Each of the following operation modes is upwards compatible to the previous ones, providing all their functions plus some additional ones:

(1) Monitoring: The puller is still controlled by the analog system; the computer can be used to collect, display, and record measured data. This operation mode is evidently essential for establishing the proper operation of the data acquisition hard- and software, and it can be used to compare the actions of both controllers.

(2) Manual Growth: The control signals for the heater(s) and the four motors are generated by the digital system. Still, they result directly from temperature and speed

setpoints, and no closed-loop diameter control is performed. The power applied to the heater(s) can be controlled in two ways: The system permits to provide three temperature setpoints, and one power limit value. The heater power output is determined by a temperature control loop while it is less than or equal to the limit value; it is set to the limit value if the temperature controller would request a greater heater power. The transition between both sub-modes is smooth and transparent to the user.

(3) Diameter Control: In this mode, the heater temperature is not only determined by its (manually entered) setpoint but also by a control loop which tries to keep the measured crystal diameter close to its corresponding setpoint. (For practical reasons, the "manual" temperature setpoint is only slightly corrected according to the diameter deviations, which results in a safer operation and gives improved control over the growth parameters.)

(4) Crucible Lift Control: The semiconductor melt in the crucible is gradually depleted while the crystal is grown. In order to maintain the solid-liquid interface at the same location within the heater, which is essential for reproducible crystal growth, the crucible has to be raised slowly during the growth run. This is done automatically in this operation mode, using a specially developed algorithm.

## 1.3 CRYSTAL GROWTH AUTOMATION

A significant improvement of the current performance of the crystal growth process, in particular, of its yield, can only be expected if it is possible to grow crystals reproducibly, with repeatable properties. This implies, however, a higher degree of process automation in order to reduce the influence of the irregularities inevitably induced by human control actions. Evidently, a digital controller is much more suitable for automating a process than the conventional analog systems. (Although the Cambridge Instruments analog controller permits to control the crystal diameter automatically over large parts of the growth process, its total operation is far from automatic, and some very crucial operator actions are still required within the "automatic" growth phase.)

The digital Czochralski Growth Control System (CGCS) was, in general, designed to duplicate the existing analog controller. This is not true, however, for the approach chosen towards process automation. Our approach is not based upon a simple control of essentially one system parameter (namely, of the crystal diameter setpoint) but on the reproduction of <u>all</u> actions pertaining to the process. However, crystal growth is a highly complex operation which is strongly influenced by unforeseeable effects like random changes in the melt flow pattern in the crucible. It was, therefore, regarded an unrealizable task to automate an entire growth run by blindly repeating a fixed pattern of actions; we felt automation could only be achieved reasonably by splitting the process into small steps which are more promising targets for automatic control. The system was, furthermore, designed to permit gradual improvements of such process steps, in order to optimize them more or less independently. The optimized steps can be joined together in a suitable way, being executed conditionally if required, to finally control an entire growth run.

The following features were therefore provided in the digital Czochralski Growth Control System in order to allow the optimization of the growth process:

(1) The system permits to modify interactively not only the actual growth data setpoints (for example, the diameter or the motor speed setpoints) but also any arbitrary internal system parameter ("Variable") which has an impact on the process. This applies specifically to the control loop parameters (e.g., to the gain of a control loop).

(2) The above changes can be made not only instantaneously but also slowly, by "ramping" a parameter linearly from its current to its desired final value within an arbitrary time. This approach prevents not only abrupt changes which are likely to upset a delicate process, it offers also a simple but efficient tool to automate process sequences. (For example, the cone between the seed and the crystal body can be grown by ramping the crystal diameter setpoint from the seed diameter to the intended crystal diameter within a time determined by the pull rate and the planned cone length.)

(3) Operator commands which affect the actual growth process can be optionally recorded on a disk file; the time at which a command was issued (relative to the start of command recording) is added as a tag to each command record. These "Macro" command files can be edited off-line, and invoked during a later growth run where they repeat exactly the recorded sequence of operator actions. Since prerecorded commands may be arbitrarily interspersed with commands entered by the operator during the run, and since the combined sequence of commands may be recorded again on

a new disk file, the system achieves a "learning" ability. This command recording makes sense for self-contained process steps only (for example, for heating up the furnace, or for starting the growth proper), but it saves the operator a number of actions which frequently have to be done within a very limited time, and it prevents the inadvertent omission of important process steps.

(4) Further process automation can be achieved by the conditional execution of such Macro command files. A pre-recorded set of commands is started only if and when a system parameter which can be arbitrarily defined with the pertinent command obtains a certain numeric relation (e.g., greater than or equal) to a given constant. Such Conditional commands may also be issued from a Macro file; it is, therefore, possible to concatenate Macro files depending on the current status of the system. Even relatively complex process steps like seeding can thus be automated.

Although the current design of the Czochralski Growth Control System does effectively permit a fully automated growth (with only one operator interaction in addition to the starting of the growth procedure), the expertise of a human operator is still required. The task of the operator is, however, reduced to supervising the process and interacting in the case of a malfunction (e.g., if the crystal "twins"). The current CGCS can not react to some events simply because it can not "see" them. Any attempt to further improve crystal growth automation must therefore be based on the introduction of additional information, especially, of data supplied by suitable optical sensors.

# 2. THE COMPUTER ENVIRONMENT OF THE CZOCHRALSKI GROWTH CONTROL SYSTEM

## 2.1 BASIC CONSIDERATIONS

The digital Czochralski Growth Control system consists essentially of two parts which are linked together relatively loosely: One part, the "brains" of the system, is a suitable microcomputer, the other part is constituted by the hardware which interfaces the digital control computer to the essentially analog outside world. We will deal with both parts separately.

Microcomputer systems for industrial applications are usually designed exactly for the control task which they have to perform, i.e., with built-in software and a dedicated interface to the operator and to the process they have to control. Frequently, they feature only a very restricted set of function keys for operator input, and limited display facilities for the output of system status and data. We felt that such a system concept would hardly meet the requirements of an experimental system which was supposed to offer the following characteristics:

* Flexibility: The system software must be easy to modify, in order to adapt the system to varying demands, to introduce new features, and, last but not least, to correct programming errors.

* Versatility: The control computer should not only be able to control growth runs but also assist in the evaluation of measured data taken during crystal growth, and permit the preparation of experiments.

* Stand-alone operation: The growth controller computer should be used as a stand-alone unit, without requiring a host system for data transfer, evaluation, and maintenance.

* Interactive operation: The system should be run in an interactive mode, permitting a dialog between the operator and the controller computer. This was regarded particularly important since the main target of the project was to learn about the dynamics of the crystal growth process, rather than producing crystals on a large scale according to pre-determined rules.

* Data display and logging facilities: As a consequence of the above considerations, it was regarded essential that

the system should be able to display, evaluate, and record as many growth related parameters as possible.

All these demands cannot be fulfilled by a dedicated computer system with completely built-in software resident in ROM (Read Only Memory). It is not only an awkward procedure to modify ROM resident programs, particularly if frequent changes are required, it is even close to impossible to accommodate lengthy and frequently conflicting routines within the limited memory space available. Since it was necessary anyhow to provide mass storage devices for growth run data logging, we planned a generic disk-based microcomputer system which permits to load arbitrary programs from flexible disks. Command input to and data output from the control computer is handled by a standard CRT terminal which permits interactive operation and data display.

## 2.2 COMPUTER HARDWARE

The hardware of the controller computer is based upon an Intel 8085 eight-bit microprocessor. This particular processor was chosen because of the vast experience we already had with it and because of the support software which was already available for it, which permitted to expect a fast system development. The experiences made with comparable applications showed that the processor's performance is sufficient if a system is well designed. The 8085 is able to address a 64 KByte memory space (plus 256 Input/Output (I/O) ports); with regard to the desired flexibility and versatility, as much of this memory space as possible was to consist of read-write memory (RAM - Random Access Memory). Only the absolute minimum of ROM which is indispensable for the operation of a computer was provided; the ROM resident code has, essentially, to control the loading of the actual application software from disk. The memory components available suggested, in addition, a memory bank switching approach which further reduces the amount of memory space consumed by ROM: The total ROM area of 16 KBytes is subdivided into two banks of equal size which can be activated alternately and which require, therefore, only 8 KBytes of address space. One bank holds confidence test routines and a Monitor which are only needed for starting and/or debugging the system; the other bank is reserved for permanently required operating system routines. Therefore, 56 KBytes are available for RAM within the 64 KBytes address space; Fig. 3 shows a memory map of the controller's computer.

The controller computer is built of commercial OEM (Original Equipment Manufacturer) components most of which are supplied by Intel Corporation; these boards are interconnected via Intel's Multibus. The system configuration is shown in Fig. 4: An Intel iSBC 80-24 Single Board Computer board holds the 8085 CPU, the 2x8 KBytes of ROM, 8 KBytes of high-speed RAM, and an Intel 8231 Numeric Processor on an iSBX 331 expansion board which permits to increase the throughput, particularly of data output to the system console. Two expansion boards, an iSBC 517 I/O, and an iSBC 028 Memory Expansion Board, provide additional I/O lines and the remaining 48 KBytes of RAM, respectively. An iSBC 204 Floppy Disk Controller board constitutes the interface to the mass storage which consists of two (industrial standard) 8" single side, single density flexible disk drives with a storage capacity of 250 KBytes each.

A standard "dumb" CRT terminal serving as an operator console is connected to the iSBC 80-24 Single Board Computer via an RS-232 serial interface. A similar serial interface on the iSBC 517 I/O Expansion Board connects to a printer whose main task is providing a hard copy of the dialog between the operator and the Czochralski Growth Control System.

## 2.3 COMPUTER-PULLER HARDWARE INTERFACE

The controller computer has to monitor and generate a number of analog and digital signals which were listed in chapter 1.2 of this documentation. The interface to the analog signals consists of one Analog-to-Digital (A/D) and one Digital-to-Analog (D/A) Converter board. Both boards are interconnected to the microcomputer proper via the Multibus system bus; data is read from and written to them via I/O port accesses.

The A/D Converter is a Data Translation DT772/5716-32DI-B-PGH board which features 32 differential input channels with a sensitivity of ±10 V (which may be increased by a factor of up to 8 under software control). The voltage of the (software selectable) input channel is converted into a 16 bit integer value by the board, corresponding to a resolution of 1/65,536; this data is read and eventually processed by the computer. A bank of isolation amplifiers between the signal sources and the A/D converter prevents ground loops which might induce noise and provides the necessary pre-amplification of low-level signals like the outputs of thermocouples.

The analog control voltages for the puller are output by a Burr-Brown MP8316-V D/A Converter board. This board provides 16 channels with an output voltage swing of ±10 V; its resolution is 12 bit (1/4,096). Eight of the 16 output channels are reserved for the interconnection to an analog chart recorder for online data output.

Digital I/O of the motor direction information and of the controller selection is performed via a series of digital I/O ports on the iSBC 517 I/O Expansion Board. These signals are buffered and pre-processed by a simple external digital circuit. Relays constitute the actual input and output interface to the puller, permitting absolute isolation between the puller's circuitry and the computer.

## 2.4 THE OPERATING SYSTEM OF THE CONTROLLER COMPUTER

A complex process like crystal growth requires basically real-time control, i.e., control operations whose chronological order is neither pre-determined nor predictable. Although there are several ways of programming such systems, an advantageous approach which, in addition, offloads the programmer from providing many standard routines is using a special real-time operating system. Such operating systems schedule the execution of parts of program code ("tasks") which are dedicated to certain actions of the controller; they also provide, in general, support of external devices like a console terminal or disk drives.

One such real-time operating system is Intel's Real Time Executive for 8080/8085 Microprocessors, iRMX-80. Although the original version of this operating system is obsolete and was withdrawn by Intel without a replacement several years ago, it was considered to offer still satisfactory performance, particularly because several of its routines have been replaced by the author with improved and more efficient ones.

Still, iRMX-80 is basically intended to support dedicated process controllers, primarily with ROM-based software, although it provides disk support and the possibility to bootload complete real-time application systems from disk. In contrast to newer real-time operating systems, iRMX-80 does not offer utility programs for the maintenance of disks and disk files (for example, for disk formatting, file copying, file display, or file listing) which would have had to be written specially for our application. Fortunately, the programming interface of iRMX-80 (i.e., the parameter passing conventions for system function calls) is very similar to the one used by Intel's System Implementation Supervisor operating system ISIS-II under which Intel's 8-bit microprocessor development systems run and for which a wealth of system utilities is available.

In order to permit the execution of such utilities on the Czochralski Growth Controller computer, a special iRMX-80 based system was written which emulates the functions of ISIS-II and which was accordingly called RXISIS-II. This operating system emulator is loaded from disk when the computer is switched on, and it is the "hub" of all system operations.

RXISIS-II permits to run various utility programs, including a BASIC interpreter and a screen editor; some programs supporting the Czochralski Growth Control System (CGCS) (e.g., the Macro Command Editor and the Data File Display program) use its services, too. Application programs are loaded into a reserved memory area; they use the system support (for example, for disk I/O) provided by RXISIS-II which remains resident within the system, and they return control eventually to RXISIS-II (Fig. 5).

This approach is fine for running relatively simple support software which does not even need real-time performance; it cannot be used, though, to execute a complex program like the CGCS. Still, the CGCS is designed to be started under RXISIS-II like any utility program, but it constitutes, in fact, a totally independent real-time application which replaces RXISIS-II totally (Fig. 6). It does not require the ISIS-II emulation since its routines interface directly to iRMX-80. Upon exit from the CGCS, RXISIS-II is re-booted and available again for the execution of support software.

iRMX-80 is a modular operating system, which permits to include only those functions into an application system which are actually required. This approach allows to keep only a "common denominator" of all perceivable application systems in ROM, meeting the requirement of a minimum size ROM resident code stated above. The ROM bank which is active during the regular operation of the system holds, therefore, the iRMX-80 "Nucleus" which is a set of housekeeping routines which are needed for any real-time system. Obviously, a Boot Loader must be included into the system ROM which permits to load either RXISIS-II or the CGCS (or any other iRMX-80 based real-time application system) from disk into RAM. Since there was still space available in the system ROM, a Terminal Handler was also stored there which controls the input and output of data from and to the console terminal. Both routines had to be written specially, replacing standard iRMX-80 software. The Boot Loader is more versatile than the standard iRMX-80 one since it can also be used to load arbitrary program code rather than complete real-time systems only, and it is by orders of magnitude faster. The Terminal Handler supports output of data to a CRT screen with a fixed format, while its standard iRMX-80 counterpart only permits continuously scrolled "Teletype" output, and it provides many more features, including printer output via a second RS-232 interface.

The second ROM bank is active only immediately after power-on, and during program debugging operations. It contains a Confidence Test and a Monitor program. Both operate totally outside the iRMX-80 environment; the real-time system is "asleep" while the Monitor is active. The Monitor permits to inspect and change the contents of memory locations and processor registers, to read from and write to I/O ports, and to execute program code up to given breakpoints. It can be entered very easily from RXISIS-II, e.g., by pressing the "Break" key of the console terminal. Since it is obviously not desirable to have the CGCS "asleep" if anybody hit the "Break" key inadvertently, this feature is locked out by the Czochralski system. (Similarly, control is vectored to the Monitor under RXISIS-II but not within the Czochralski system if a fatal disk error condition is detected.)

The Confidence Test can be accessed from the Monitor; it permits to check the integrity of the system hardware, including the console terminal, the printer, and the two disk drives. Its memory test portion is, in addition, executed after each power-up.

For further information about iRMX-80, the Alternative Loader Task, the Alternative Terminal Handler, and RXISIS-II with its Monitor and Confidence Test routines, please refer to the pertinent documentations listed in Appendix A.

# 3. THE CZOCHRALSKI GROWTH CONTROL SYSTEM

The Czochralski Growth Control System (CGCS) is invoked from RXISIS-II, and it is in some respects an extension of the RXISIS-II functions. Some special measures have been taken, though, to guarantee a proper operation even in the case of failures. While, for example, a disk error constitutes a fatal situation under RXISIS-II, it is only reported in the CGCS but does not affect its overall operation.

## 3.1 STARTING THE CZOCHRALSKI GROWTH CONTROL SYSTEM

NOTE: The system needs the CGCS system disk permanently in drive 0. The operator must by no means exchange this disk unless prompted to do so (see the EXCHANGE command). To be save in the (improbable) case of a disk error on the system disk, a second system disk should be kept at hand which must, however, be of the same system version. The system will crash inevitably at the attempt to install a disk in drive 0 which holds a different CGCS version!

The CGCS is invoked from RXISIS-II like any other RXISIS-II function, namely, via a call by its name, CZOCHR. Provided the disk in drive 0 holds a valid copy of the CGCS, it will be loaded, and a sign-on message is displayed. During this initialization, the system checks whether the A/D Converter hardware is installed and operational, and it enters into a Test mode if this is not the case. An information message "Test Run" is displayed in this case, and input from the A/D converter and output to the D/A board are suppressed. This feature permits testing of the CGCS software in an environment which does not provide the hardware interface to the puller; running the CGCS with disabled inputs allows, in addition, to simulate input parameters for testing purposes. (Analog I/O can also be suppressed under software control in a fully equipped hardware environment; compare chapters 3.6 and 4.4.3.)

Among may other initialization chores, the CGCS disables the BREAK key on the console terminal, and enforces a duplication of Monitor output on the printer. All inadvertent entries into the Monitor program will therefore show up in a Documentation printout.

During the entire growth run, the CGCS checks the integrity of its program code periodically. RXISIS-II should be re-booted, and the CGCS re-started as soon as possible if a memory error is reported in order to avoid unforeseeable reactions of the system.

Subsequently, the CGCS prompts for the current date (which is not updated even if a run extends beyond midnight) and time, and for an arbitrary run identification code. Date and time must be entered in the format displayed by the CGCS; the seconds can, however, be omitted (they will be assumed to be zero in this case). The operator can accept or reject these entries. A plain "Return" in response to the confirmation prompt will accept the data displayed in the top line of the screen.

The permanent display screen which comes up after the above procedure holds the following items:

(1) Date, time, and run identification in the top line. Two times are displayed, namely, the actual time (in 24 hours format), and an internal system time which starts at zero when the system is initialized, and can count up to 95 hours, 59 minutes, and 59 seconds. (It wraps around to zero after 96 hours, and starts counting up again.) The top line holds, in addition, a space between the run identification and the system time where the name of a Macro Command will be displayed while it is executed.

(2) A regularly updated display of measured system parameters and of setpoints. Two columns are provided for the display of the setpoints: The left column holds the currently valid setpoint, whereas the right column displays the final setpoint which differs from the current setpoint if a parameter is being ramped. In the case of a controlled parameter, the right column shows a setpoint input to the controller. Parameters which can be entered as setpoints are, in addition to their full names, identified by the two character abbreviation which is required by the SET or CHANGE commands. The system was designed to accommodate a three-zone heater. Therefore, three heater temperatures and three pairs of power values are displayed. (Currently, only the first set of data is meaningful; the measured data for the second and the third channel have been tied to those of the first channel.) There are two output power values for each channel, referred to as "In" and "Out"; the "In" values specify the percentage of maximum power which is input to the power controller, while "Out" gives the actual output power; both are scaled to lie between 0 and 100. (The "In" values were, in fact, calculated and output by the CGCS, whereas the "Out" data are measured data input by the CGCS. "In" and "Out" refer to the power controller, not to the CGCS.)

(3) Internal system status information: This information comprises the number of parameters being ramped, and the number of Conditional Macro commands pending, against their respective maximum values (20 and 8, respectively). Furthermore, the operation mode (see MODE command) is displayed close to these two values in the top right corner of the screen.

(4) Command echoes and system messages: While the remainder of the CGCS output screen is in a fixed format and updated in a random access mode, the echo and message area (five lines in the bottom third of the display) is scrolled up as information is added in the bottom line. The echoes of operator entries are displayed there, and messages issued by the system are directed there, too. In addition, the same area is used by some commands for the display of menus or auxiliary information. The scrolled portion shrinks to four lines if auxiliary data display is requested with the DEBUG Continuously command. In this case, the top line of the scrolled portion is used for the DEBUG output.

(5) Command prompt line: All operator actions are requested in the last line but one on the screen.

(6) Input area: The bottom line is reserved for building a command line. The same rules apply to the entry and to the editing of commands which were specified for RXISIS-II.

During the initialization of the system, some commands are automatically performed by the CGCS, thus saving the operator typing and making sure that all required information is entered. The system permits to open a Documentation output file (otherwise done with the DOCUMENTATION command), and requests a set of constants (see the INITIALIZATION command). Finally, the Command Interpreter's prompt "Please command:" is displayed, and the CGCS enters its regular operation mode.

## 3.2 RUNNING THE CZOCHRALSKI GROWTH CONTROL SYSTEM - COMMANDS

### 3.2.1 GENERAL REMARKS

The operation of the CGCS is determined by independent commands which are interpreted by the Command Interpreter (one of the CGCS's system tasks). There are two types of commands, namely, the Internal, and the Macro commands. Internal commands are built right into the program; they provide the basic control functions. Macro commands, in contrast, are in fact disk files which are read when their name was detected as a Macro command. These disk files hold, in turn, one or more Internal commands, with a time information attached. These Internal commands are therefore not only executed in the order in which they were recorded on the file but also with the same timing. Macro command files can be generated either by directly recording the commands entered on the console during a growth run, or with the Macro Command Editor COMMED.

Internal commands are generally executed in an interactive way, i.e., the operator is prompted for further information if necessary. Some of the commands permit the entry of all information required in one single command line, which shortens the dialog between the system and the operator significantly. All items which may be entered together with the command keyword are specified in the summary of Internal commands in chapter 3.2.2. Commands which are likely to affect the operation of the system significantly require, in general, a reconfirmation of the data entered by the operator. In most cases, the operator can accept the data shown to him by entering "Y(es)"; any other entry, including an empty line ("Return" only), cancels the command.

All valid Internal commands and the descriptions of their purposes are listed below in alphabetical order, first in a short, and then, in a comprehensive list. It is not possible, however, to give a similar list of Macro commands since they may be freely defined at any time. It is therefore up to the operator to keep a record of his Macro commands and of their functions. The following syntax is used:

CAPITALS constitute the part(s) of the command which must be entered exactly as specified.

lowercase parts of the command keyword are optional. They are specified here for clarity and may also be typed in but are ignored by the Command Interpreter.

Items in angular brackets < > have to be replaced by the appropriate contents, e.g., a parameter value or a Macro command name.

Items in square brackets [ ] are optional and may be omitted.

Items included in braces { } and separated by a vertical bar | are optional but one item of the list must be specified.

Items must be separated by at least one space (except within file names).

Note: Commands may be issued in arbitrary order. A command is, however, only recognized when the prompt "Please command:" is displayed!

## 3.2.2 SUMMARY OF INTERNAL COMMANDS

```
CALCulate [{R|I|H}]
CHANge [{D|Tn|SL|CL|SR|CR|PL|<varname>}[<value> [<time>]]]
CLEAr [<varname>]
COMMENT [<arbitrary text>]
DATA
DEBug [C [<varname> [{1|2|3|4}]]]
DEBug [C [<hexaddr> [{A|I1|I2|R|H1|H2|H4} [{1|2|3|4}]]]]
DEBug [D [{<varname>|<hexaddr>}]]
DEBug [M [<varname>]]
DEBug [M [<hexaddr> [{A|I1|I2|R|H1|H2|H4}]]]
DEBug [O [{1|2|3|4}]]
DEBug [R [{<varname>|<hexaddr>}]]
DEBug [S [{<varname>|<hexaddr>}]]
DIRectory [{0|1}]
DISPlay [<varname>]
DOCUmentation
DUMP
END
EXCHange [{0|1}]
EXIT
FILEs
HELP or ?
IF [<varname> [{<|=|>}[{<|=|>}] [<value> [<macro>]]]]
INITialize
MODE
PLOT [{<varname>|<hexaddr>} [{1|2|3|4|5|6|7|8}]]
QUIT
RESEt [<initial weight> <initial length>]
RESTore
SET [{D|Tn|SL|CL|SR|CR|PL|<varname>} [<value> [<time>]]]
STARt
```

## 3.2.3 COMPREHENSIVE DESCRIPTION OF THE INTERNAL COMMANDS

CALCULATE: This command permits to calculate the sum, the difference, the product, and the quotient of two numbers. The input format and the treatment of the numbers depends on a switch entered with the command: The switch is "R" for floating-point ("REAL") numbers, "I" for integers (which must lie between -32768 and 32767), and "H" for hexadecimal values (e.g., memory addresses) which have the same numeric range as integers. Input values are explicitly requested in any case. The result is displayed in decimal and hexadecimal form, with the internally used hexadecimal format for floating-point numbers if applicable.

CHANGE: This command permits to modify the value of one of the nine primary system setpoints (crystal diameter, three heater temperatures, seed and crucible lift and rotation speeds, and power limit), or of an arbitrary system Variable (see chapter 3.6 and Appendix F). CHANGE determines the current value of the specified parameter and adds the input value to it, thus permitting relative changes. Since the actual execution of the command is kept separate from the operator interface, the actual value of the target parameter may differ from the one displayed during the processing of the command if the target parameter is being ramped when the command is issued. Setpoints which are used as an input to a controller (e.g., the Temperature setpoints in Diameter controlled modes), are displayed with the values output by the controller. CHANGE permits a smooth transition of the parameter between its current and its final values by allowing a transition time during which the parameter is ramped (see remarks about parameter ramping in chapter 3.3). The transition time may range from zero to 9999 minutes (in fact, longer transition times are possible but cannot be displayed any more). The shortest non-zero transition time is one second; this value is used for all non-zero transition time values less than one second (0.017 minutes). The CHANGE command may be completely entered in one line, or in any combination of items. It may be recorded to and executed from a Macro command file.

CLEAR: The command CLEAR removes pending Conditional Macro commands from the Conditional Command queue. It may be used to branch between Macro commands if a condition specified with an IF command is not met within a given time. There are two types of CLEAR commands: An unconditional CLEAR which removes all pending Conditional Macro commands, and a Selective CLEAR which cancels only those Conditional commands which refer to the Variable specified with the CLEAR command. CLEAR can be recorded to and executed from Macro command files.

COMMENT: This command inserts one line of comment into the Data output file. The comment line is tagged with the operation mode, time, and length grown information and embedded between the (binary) records in the Data file, thus permitting the correlation between arbitrary events and the data recorded. Even if no Data file is in use, the comment line is recorded in the Documentation output. (In fact, the COMMENT command is the only one to provide arbitrary text in the Documentation output.)

DATA: The DATA command permits to open or close the Data output file. It offers the operator to open a Data file if there is no open such file, and it permits to close the Data file if it is invoked while a Data file is open. After a disk error, the file which was involved in the error is flagged as "inactive". Not reactivating an inactive file is equivalent to closing it. The functions of DATA may be also accessed through the FILES command.

DEBUG Continuously: One member of the DEBUG command group, the DEBUG Continuously command permits the continuous display of the values of up to four system Variables. The data output provided is updated at the same rate as the fixed screen output (once every five to six seconds). (In fact, the memory locations specified with DEBUG Continuously are sampled once every second; their values are also recorded in the Data file.) Data can be selected for display either by specifying a Variable name, or by submitting the hexadecimal address of the memory location(s) whose contents are to be displayed. In the latter case, an additional format information is required since DEBUG does not know what kind of data resides at an arbitrary storage location in memory. The display formats available are ASCII (A), interpreting one byte at the specified address as a (printable) character, one and two byte decimal integers (I1 and I2, respectively), decimal floating-point (REAL - R), and one, two, and four byte hexadecimal representation (H1, H2, H4). Finally, one of the four DEBUG output channels (numbered 1 to 4) must be specified to which the output is to be directed. (Channels 1 to 4 are displayed in the DEBUG output line on the console from left to right.) The DEBUG Continuously command may be completely entered in one line, or in any combination of items. It may be recorded to and executed from a Macro command file.

<u>DEBUG Display:</u> The DEBUG Display command displays the contents of one or several adjacent memory locations which have been specified either by a Variable name, or by a hexadecimal address. (For displaying the contents of a Variable in its standard representation, the DISPLAY command is probably more convenient.) The four bytes starting at the given address (or part of them) are displayed as ASCII characters, in hexadecimal notation, as one and two byte decimal integers, and as (four byte) floating-point numbers. The command may be completely entered in one line, or in any combination of items.

<u>DEBUG Modify:</u> This command permits to modify one to four bytes in memory whose starting address must be specified either with a Variable name, or as a hexadecimal number. The program knows how many bytes have to be modified to change the value of a Variable specified by name, but the data format has to be submitted separately if a hexadecimal address is used. The formats available are ASCII (A), interpreting one byte at the specified address as a (printable) character, one and two byte decimal integers (I1 and I2, respectively), decimal floating-point (REAL - R), and one, two, and four byte hexadecimal representation (H1, H2, and H4). The program displays the current contents of the specified location(s), and prompts explicitly for a new input value. With the exception of the new value, the entire command or parts of it can be entered in one command line. (For changing Variables specified by name, the SET and CHANGE commands are probably more convenient; in addition, they offer the ramping feature which is not supported by DEBUG.) The DEBUG Modify command can be recorded to and executed from a Macro command file.

<u>DEBUG Off:</u> While DEBUG Continuously turns on the output of Debug data, DEBUG Off turns it off again. The location (1 to 4) which is to be turned off must be specified. The command may be entered in one or in two lines. It may be recorded to and executed from a Macro command file.

<u>DEBUG Resume:</u> This command affects the internal operation of the system. It should only be used for debugging purposes. Therefore, no further information is given here.

<u>DEBUG Suspend:</u> This command affects the internal operation of the system. It should only be used for debugging purposes. Inconsiderate use of this command may disable the

CGCS entirely. Therefore, no further information is given here.

DIRECTORY: The DIRECTORY command displays the contents of the directory of the specified disk. In addition to the file names, the disk label and the numbers of sectors in use and free on the disk are displayed. Note: The actual number of sectors in use may be much greater if a file is open for output on the specified disk. The actual number of used sectors cannot be determined, though, since it is an internal parameter of the operating system. The numbers displayed for the used and free sectors are, however, preceded by a ">" and a "<" sign, respectively, in this case. The command may be entered in one line.

DISPLAY: This function displays the value of a Variable submitted as a parameter with the call. The command may be entered in one line.

DOCUMENTATION: A call to DOCUMENTATION permits to switch on or off the Documentation output on the printer or on a disk file. DOCUMENTATION offers to open a Print file if no such file is open, and to close it if it is open. During the file opening procedure, DOCUMENTATION permits to set the interval between Data Dumps to the Documentation output (compare command DUMP). Any arbitrary interval between 1 and 255 minutes may be specified; periodic Data Dumps may be disabled altogether. After a disk error, the file which was involved in the error is flagged as "inactive". Not reactivating an inactive file is equivalent to closing it. The DOCUMENTATION routine is automatically invoked when the system is started; it may also be accessed from the FILES command.

DUMP: This command initiates a dump of 21 system parameters (essentially, of the measured data) to the Documentation output. In addition, it triggers one record written to the Data file.

END: The END command is the official way to terminate a command record in the Control Output file (which eventually may be used as a Macro command file). Although no more entries are added to the Control Output file after an END command, the file remains open, and the next record may be started at any time with a START command. (This permits to use one Control Output file throughout a growth run to which certain command sequences are recorded; the records in it can be separated into several Macro command files using the Macro Command Editor. Note, however, that an END command preempts a Macro command file used for input regardless of whether there are more commands after the END command or not.)

EXCHANGE: This command permits to exchange a defective or full disk safely. It closes the files on the specified disk which are still open, prompts the operator to substitute a new disk, and re-opens all files on the new disk which were open on the old disk when the operator indicated to the system that the new disk was installed. Since the output files are opened with the same names on the new disk, any file with an identical name on the new disk is overwritten. In addition, the output files need some editing because control structures used on the Data and Control Output files are not provided by EXCHANGE. (It is sufficient to concatenate the two output files with the ISIS-II/RXISIS-II COPY command, or to concatenate the second part of a Data or Macro command file with a separately generated file header.) Note that a Macro command will be preempted which is being read from a disk which is to be EXCHANGEd.

EXIT: The only regular way to leave the CGCS is the EXIT command. Depending on the current operation mode, the EXIT command "cleans up" the controller. It stops the lift motors if the puller is under the control of the CGCS, reduces the heater power to zero within six hours (unless the power is already zero), stops the rotations, and relinquishes, finally, control to the analog controller. Several safety procedures prevent the accidental execution of this function.

FILES: This command displays the current status of the Print, Data, and Control Output files and their names if the files are open. Subsequently, it permits to open or close one of the three files, entering the respective DOCUMENTATION, DATA, and Control Output file handling routines. After a disk error, the file which was involved in the error is flagged as "inactive". Not reactivating an inactive file is equivalent to closing it.

<u>HELP:</u> The HELP command (or, alternatively, a simple question mark ("?")) provides a set of command menus on the screen. The menus displayed comprise a summary of the Internal commands, the currently available Macro commands, and an extensive explanation of each command. The Macro command list and/or the extensive help display may be skipped if not needed.

<u>IF:</u> This command permits the conditional execution of a Macro command (it does not work with Internal commands). The Macro command specified with the IF call is executed if and when a condition is met which is based on the numeric relation between a Variable and a constant which are submitted as parameters of the IF call. The numeric relations may be "greater than" (">"), "equal to" ("="), "less than" ("<"), or any combination of two of these three ("<>" stands for "not equal"). The order of the relation characters does not matter; "=>" is identical to ">=" and means "greater than or equal to". Eight (8) Conditional Macro commands may be pending at a time; any Conditional command issued while the maximum number of commands are pending is ignored, and a pertinent error message is displayed. The command may be completely entered in one line, or in any combination of items. It may be recorded to and executed from a Macro command file.

<u>INITIALIZE:</u> This command permits to assign values to certain system parameters which cannot be (easily) changed otherwise since they are kept in memory in a pre-processed form to facilitate control operations. The values set with INITIALIZE are the diameters of the crucible and the seed, the amount of boric oxide used, and the densities of the solid crystal, the crystal melt, and the boric oxide melt. Since these values are, in most cases, hardware dependent constants anyhow, INITIALIZE offers default values which can be accepted with a plain "Return", or overwritten by new data. INITIALIZE is automatically executed when the system is started; it must be called during a growth run when the crystal is melted back partly, and growth is resumed with a full-diameter crystal within the boric oxide melt. In this case, the diameter of the crystal must be specified as a seed diameter, in order to provide a correct diameter evaluation after a subsequent RESET call.

<u>MODE:</u> The MODE command permits to select one of five operation modes which are numbered 0 through 4. Each mode is a

superset of the functions of the preceding one. Mode 0 provides monitoring without control, Mode 1, a basic (manual) control but no diameter control. The latter is possible with Mode 2 which, however, does not include an anomaly compensation. Mode 3 provides anomaly compensation, and Mode 4, in addition, a Crucible Lift control which is based on the exact amount of melt withdrawn from the crucible during the crystal growth. Each mode change is reported by the system, and an automatic Data Dump is triggered. The MODE command may be recorded to and executed from a Macro command file.

PLOT: The PLOT command permits to output continuously (similar to the DEBUG Continuously command) the values of p to eight locations in memory which can be specified by Variable names or by absolute hexadecimal addresses. While DEBUG Continuously routes its output to the operator console and the Data file, the PLOT output is directed to eight spare channels of the D/A converter which are connected to a suitable chart recorder. PLOT can only handle Variables which are in INTEGER*2 notation, which applies to all measured parameters and control output signals, and to a number of internal system parameters (compare chapter 3.5 and Appendix F). A number of auxiliary locations were provided which hold "expanded" values of parameters of which only a narrow numeric range is of interest. For further information on the PLOT command, refer to chapter 3.5. The PLOT command may be recorded to and executed from a Macro command file.

QUIT: The QUIT command permits to preempt a currently active Macro command.

RESET: The proper operation of the diameter evaluation routines requires a RESET command at the beginning of the actual growth. The RESET command resets the length grown counter and the weight output to zero or to values specified with the call, and initializes the internal data structures of the diameter routines. It is indispensable to issue such a command after each INITIALIZE command (including the one automatically performed at the beginning of the CGCS operations), and after each irrecoverable "Speed overflow" error, when the puller is again in a well-controlled condition and growth can resume. (Otherwise, no new diameter output is generated, and diameter control is not possible.) A RESET command which sets the crystal length and weight to zero is automatically generated if necessary when the operation mode is changed to one of the diameter controlled ones (Mode 2 through 4). It is possible to maintain the current length and weight values with a RESET command either by answering the pertinent questions accordingly if in the interactive mode, or by specifying a value for the parameter to be maintained which is less than twice its most negative value (i.e., less than -16000 for the crystal weight, and less than -1200 for the crystal length). The RESET command may be recorded to and executed from a Macro command file.

<u>RESTORE:</u> The RESTORE command restores the console output if it was corrugated, which can happen very easily if one of the function keys on the console terminal is pressed inadvertently, or if the "Return" key is pressed while the cursor is in the bottom line of the screen, e.g., after the entry of a full input line of 80 characters. It does not affect the actual control operations of the CGCS.

<u>SET:</u> This command permits to modify the value of one of the nine primary system setpoints (crystal diameter, three heater temperatures, seed and crucible lift and rotation speeds, and power limit), or of an arbitrary system Variable (see chapter 3.6 and Appendix F). It sets the specified parameter to the input value, thus permitting absolute changes. SET permits a smooth transition of the parameter between its current and final values by allowing a transition time during which the parameter is ramped (see remarks about parameter ramping in chapter 3.3). The transition time may range from zero to 9999 minutes (in fact, longer transition times are possible but cannot be displayed any more). The shortest non-zero transition time is one second; this value is used for all non-zero transition time values less than one second (0.017 minutes). The command may be completely entered in one line, or in any combination of items. It may be recorded to and executed from a Macro command file.

<u>START:</u> This commands starts the recording of commands in the Control Output file. If no such file is open, START permits to specify and open a Control Output file. Command times recorded in the output file are relative to the time of the START command. (For example, a SET command issued 35 seconds after the START command will be executed 35 seconds after the Control Output file was invoked as a Macro command during a later run.)

## 3.3 PARAMETER RAMPING

Parameters entered with the SET and CHANGE commands may be ramped linearly between their current values and the final values specified with SET or CHANGE. Arbitrary ramping times between 1 second and 9999 minutes may be used. Up to 20 parameters (primary system setpoints or arbitrary Variables) may be ramped at a time, no matter whether the pertinent commands were entered from the console, or from a Macro command file. The number of parameters which are currently ramped is displayed on the console screen. Note: A SET or CHANGE command requesting parameter ramping which is issued when already 20 parameters are being ramped will be executed instantaneously, without ramping. Watch therefore the number of ramped parameters carefully when you use extended ramping and/or Macro commands. A SET or CHANGE command referring to a parameter which is already being ramped does not increase the number of ramped commands. Parameter ramping can be halted by commanding CHANGE <parameter> 0 0 (change the parameter by 0 within 0 minutes).

## 3.4 MACRO COMMANDS

All operator entries input when the "Please command:" prompt is displayed are first compared to the list of the above Internal commands. If no match is found between the first four characters of the operator input and any one of the Internal command names, the CGCS assumes that a Macro command was requested, and searches the system disk in drive 0 for a file with an extension ".CMD" whose name matches the operator entry. Therefore, the following rules apply to Macro command names:

(1) Macro command names may consist of one to six alphanumeric characters; the first character must be alphabetic.

(2) The first four characters of the Macro command (three characters if the command begins with "DEB") must not match any internal command name. (Note, though, that commands whose keywords are shorter than four characters have their names padded to the right with spaces. The name "SETPNT" is therefore a perfectly legal Macro name.) Macro names which are part of a Conditional command are excepted from these restrictions.

(3) A file with the name <macro>.CMD must exist on the disk in drive 0, and it must be in the special Macro command format.

(4) Macro commands generally do not take any parameters.

If any one of the above conditions is not met, an "Illegal command" message is issued by the Command Interpreter, and the command is ignored.

Macro command files comprise a set of recordable internal commands which are stored in a binary encoded format in order to save disk space and processing time. Since references to Variables are stored as the absolute binary addresses of these Variables and since Variable locations may change when software modifications are made, it is essential that Macro commands referring to absolute memory locations are only executed under the program version for which they were generated. A warning is issued if the user attempts to execute a Macro command which was designed for or generated by a CGCS version different from the one in use, and all Internal commands within the Macro command file which refer to absolute memory locations are dropped. (They are indicated to the operator, though, with an appropriate error message.) Macro command files generated under a previous system version have to be converted with the Macro Command Editor COMMED into a valid Macro command for the current system version.

Macro command files can be created in either of two ways:

(a) By recording actual commands during a growth run, using a Control Output file and the START and END commands, or

(b) With the Macro Command Editor COMMED which can also be used to modify command files recorded during a growth run.

The following Internal commands can be recorded on and later executed from a Macro command file:

```
CHANGE
CLEAR
DEBUG CONTINUOUSLY
DEBUG MODIFY
DEBUG OFF
DEBUG RESUME
DEBUG SUSPEND
END
IF
MODE
PLOT
RESET
SET
```

Macro commands can be invoked from a Macro command file, but they are not recorded in a Control Output file. This was done on purpose since a Macro command invoked from another Macro command preempts the command file from which it was invoked. (There can be only one Macro command file in use at a given time.) A Control Output file generated during a growth run receives commands issued by the operator as well as commands stemming from a Macro, and it is not possible to distinguish between both. The operator generated commands interspersed with the commands originating from the Macro would, however, be effectively lost if the Macro call were also recorded in the Control Output file. Replaying this Control Output file as a Macro file at a later stage would simply result in the Macro being preempted by the one which was invoked during the recorded run, and only the commands on the new Macro would be executed automatically. This would deteriorate the self-learning ability of the CGCS considerably.

Note: Commands issued by a Macro command file remain active even after the Macro was terminated or preempted!

## 3.5 DISK FILES

Besides the Macro command (input) files, there are three files available for output from the CGCS under the operator's discretion.

PRINT FILE: The Print file receives the complete dialog between the operator and the system. Each line of output is tagged with the absolute and the system times; the date on which the run was started and the run identification are contained in page header lines. The Print file can be opened (activated) or closed (deactivated) with the DOCUMENTATION command or via FILES. Print file output can alternatively be sent to the line printer (which is indicated by ":LP:" in the FILES display), or to a disk file. Arbitrary (valid) file names and extensions may be chosen, and the file can be opened on either disk drive. (It is recommended, though, that drive 1 is used for the Print file output because the Print file tends to become very bulky, and there is not too much room left on the system disk.) In addition to the operator dialog, Data Dumps are recorded in the Print file which contain the following items:

* Measured values of the three heater temperatures.
* Heater power input and output values.
* Measured motor speeds.

* Seed and crucible positions.
* Crystal length and diameter.
* Weight and differential weight.
* Base temperature.
* Gas pressure.

In order to conserve space, the output items are identified only with two-character mnemonics:

| | |
|---|---|
| T1 ... | Heater #1 Temperature (in millivolts) |
| T2 ... | Heater #2 Temperature (in millivolts) |
| T3 ... | Heater #3 Temperature (in millivolts) |
| SL ... | Seed Lift Speed (in millimeters/hour) |
| CL ... | Crucible Lift Speed (in millimeters/hour) |
| L ... | Length Grown (in millimeters) |
| D ... | (Calculated) Diameter (in millimeters) |
| P1i .. | Demanded Power (Input) for Heater #1 (percent) |
| P2i .. | Demanded Power (Input) for Heater #2 (percent) |
| P3i .. | Demanded Power (Input) for Heater #3 (percent) |
| SR ... | Seed Rotation Speed (in RPM) |
| CR ... | Crucible Rotation Speed (in RPM) |
| W ... | Crystal Weight (in grams) |
| DW ... | Differential Weight (in grams/minute) |
| P1o .. | Actual Power (Output) of Heater #1 (in percent) |
| P2o .. | Actual Power (Output) of Heater #2 (in percent) |
| P3o .. | Actual Power (Output) of Heater #3 (in percent) |
| SP ... | Seed Position (in millimeters) |
| CP ... | Crucible Position (in millimeters) |
| BT ... | Base Temperature (in millivolts) |
| GP ... | Gas Pressure (in PSI) |

Data Dumps are initiated in the following cases:

* Upon a DUMP command.
* At a change of the system's operation mode.
* Periodically with a specifiable interval.

In the first two cases, a Data record is also written to the Data file.

DATA FILE: All important system parameters can be recorded on the Data disk file. A set of data is compiled in regular intervals and written to disk. With regard to execution time and disk space requirements, these records are written in a not directly legible binary format; special support software which can decode Data files and output selected channels, for instance, to a chart recorder, is required. The following items are contained in each data record:

Operation Mode
System Time
Length Grown
Measured Data (17 channels - all data displayed permanently)
Auxiliary Analog Data (8 channels)
Power Output (3 channels)
Current Setpoints (9 channels - all data displayed permanently)
Auxiliary Setpoints (9 channels, as above)
Debug Continuously Addresses and Data (4 * 3 channels)
Diameter
Debug Continuously Variable types (1 channel)

Each channel holds two bytes of data; one record of 64 channels (63 active, 1 spare) fills exactly one sector on the output disk.

The Data file can be opened (activated) or closed (deactivated) with the DATA command, or via FILES. Arbitrary (valid) file names and extensions may be chosen, and the file can be opened on either disk drive. (It is recommended, though, that drive 1 is used for the Data file output because the Data file tends to become very bulky, and there is not too much room left on the system disk.) The operator has to specify an interval for the data acquisition when you open a Data file; there are about 1800 sectors available on an empty disk, and each record consumes one sector. (The remainder of the sectors on the disk is required for housekeeping.) Since it should make sense not only to record data but also to process it later on, it is probably a good idea to restrict data recording to processes which are actually of interest, and to choose the recording interval according to the dynamic behavior of the processes involved. (Once a Data file has been opened the interval can not be changed any more. A new Data file has to be opened if a different recording interval is needed.)

CONTROL OUTPUT FILE: All recordable commands (compare chapter 3.4) are recorded in a Control Output file if such a file is open, and if the START command has been issued. A Control Output file can be opened with the START command, and it can be opened and closed with FILES. The file may be opened on either disk drive, but it must be opened on drive 0 if it should serve as a Macro command file within the same run. No file name extension is required with the Control Output file name; the CGCS appends automatically ".CMD". Command recording can be deactivated with an END command at any time after a START command; the Control Output file remains open, though, until it is either closed with FILES, or until the CGCS is EXITed. One Control Output file can hold multiple Macro command records on the Control Output file which are started and terminated with the START and END commands, but the file requires editing in this case (with COMMED) before all these Macro command records can be used. (Otherwise, the first END recorded would preempt the Macro command, and all following commands would be ignored.)

Note: During a growth run, a Macro command file can be created for "instant use" in the following way:

(1) Open a Control Output file on drive 0 (important!) with an arbitrary name, preferably using the START command.

(2) Enter the command(s) you want to have in the file but be careful that you do not interfere with a growth run in progress.

(3) Close the Control Output file (with FILES), and

(4) Use it as a Macro command when required.

A Control Output file must be closed before it can be invoked it as a Macro file.

PLOT OUTPUT: In contrast to the above three output files, Plot Output is directed to an analog rather than a digital device, namely, to a multi-channel chart recorder. In general, any Variable whose type is INTEGER*2 can thus be submitted to the chart recorder output, and so can any arbitrary two-byte memory location which is referred to by its address. This includes all measured input data (which are in INTEGER*2 format anyhow), plus a number of internal system parameters. (Refer to the list of Variables in Appendix F to find the Variables which might be of interest.) In general, the absolute values of the Variables specified are output on the eight spare analog output channels, scaled from 0 to 10 V for the full range of 0 through 32767 covered by positive INTEGER*2 numbers. A message is output on the console and recorded in the Documentation output whenever a Variable changes its sign.

(Initially, all outputs are supposed to refer to positive values.)

In addition to the standard INTEGER*2 Variables, the following Variables obtained from a special treatment of internal data were provided for chart recorder output:

(1) Heater and Base Temperatures: Four Variables, EXTMP1, EXTMP2, EXTMP3, and EXTMPB, hold an expanded Heater or Base Temperature value. The full range (0 to 10 V) of the output obtained from these Variables is determined by the Variables RANGT1, RANGT2, RANGT3, and RANGTB, respectively, starting from an offset value which is set by the Variables OFFST1, OFFST2, OFFST3, and OFFSTB. Like all other Variables, these parameters can be modified with the standard SET, CHANGE, or DEBUG Modify commands; their values must be specified in millivolts. In order to PLOT on the Chart Recorder Channel 3 the temperature of the Heater 1 which is supposed to lie, say, between 22.5 and 24.5 mV, the following commands may be used:

```
SET OFFST1 22.5 0
SET RANGT1 2 0
PLOT EXTMP1 3
```

Temperature values below the specified offset will result in a zero output, and values greater than the offset plus range values, in an output voltage of 10 V. Note that the offset may be ramped, too; this permits to record a deviation from a given setpoint.

(2) Growth Rate: An expanded Growth Rate value is kept in GRRATE. A zero output corresponds to a growth rate of zero (as calculated by the Diameter Evaluation routine SHAPE); the maximum output is reached for a growth rate of 20 mm/hr. GRRATE can assume positive and negative values (the latter during meltback).

(3) Diameter Error: The Variable DIAERR holds the difference between the Diameter setpoint and the actual diameter. A zero difference is output as mid-scale (5 V); zero and maximum output correspond to an actual diameter 10 mm smaller and greater than the setpoint, respectively. Greater deviations than 10 mm result in the proper minimum or maximum output signals.

(4) Crucible Position Error: Similarly, the Variable CRPERR is set to a value corresponding to the deviation of the actual crucible position from the calcul-

ated value. A zero error is again represented as mid-scale; the maximum deviation which can be resolved is ± 10 mm. (The crucible is too low if the output is less than mid-scale.)

Any PLOT channel can be activated by the command

```
PLOT <varname> <channel #> or
PLOT <hexaddr> <channel #>
```

The command may be entered in one line, or one item at a time as requested by the CGCS. The system checks whether the type of the Variable specified is indeed INTEGER*2 (it assumes INTEGER*2 locations if a hexadecimal address was entered), and attaches the value of the specified location to the proper output channel. Channel numbers 1 through 8 are permitted. An output channel remains active and connected to a Variable until it is re-assigned; output may be de-activated with the

```
PLOT ZERO <channel #>
```

command. The analog output is updated periodically once every second.

## 3.6 VARIABLES

### 3.6.1 GENERAL REMARKS

The concept of the CGCS permits an easy way to modify any arbitrary parameter used by the system, a way which is certainly more convenient and safer than using the parameter's absolute address in memory: A virtually unlimited number of parameters can be accessed by a name unique to each parameter. The CGCS looks up the actual address and the type of a specified Variable in a directory file; the number of parameters accessible in this way is only limited by the reasonably obtainable size of this file. The directory file has the name CZONAM.Vmn, with m and n, the major and minor version code numbers. It contains Variable names, addresses, and types in a binary encoded form, and is generated from a source file VARADD.SRC by means of a dedicated program CONVAD. The directory file must be updated for each new system version since the Variables listed in it may have changed their addresses due to program modifications.

Variable names must consist of one to six alphanumeric characters; the first character must be alphabetic. Variables can either be simple storage locations, or arrays. Elements of arrays must be specified by the number of the element (beginning with 1), in parentheses immediately following the array name. (There must not be a space between the name and the opening parenthesis.) An omitted array element number defaults to 1. Valid Variable names are, for example, "TIME" or "ANAPAR(6)". The name may be entered in upper- or lowercase characters.

Chapter 3.6.2 provides a list of special Variables which are more than a simple parameter since their values directly determine the operation of the CGCS. A table of the most important Variable names, sorted according to their meanings, and a complete list of all Variables used by the CGCS are provided in Appendix F.

## 3.6.2 SPECIAL VARIABLES

System Control:

TEST This Variable puts the CGCS into a Test mode if it is set to -1; all other values maintain the regular operation of the system. In Test mode, input from the A/D converter and output to the D/A converter and the relays board are inhibited. This permits to safely assign values to an array of Variables which are otherwise set by the A/D converter's output, and to run the system with these faked "measured data" for testing purposes. (The names of the input array Variables are made up from the letter "M" plus a five character mnemonic; compare Appendix F.) Note: TEST must not be set to -1 while the CGCS is actually controlling the puller!

DIASTA This is an internal status parameter of the Diameter Evaluation routines. It may be set to -2 at the end of a growth run in order to disable the diameter evaluation and, in particular, the generation of error messages which may be triggered by some of the actions usually involved in the close-down procedure of the puller. Diameter evaluation may be enabled again with a RESET command.

ALPHA The parameter ALPHA determines the diameter evaluation algorithms within two extreme approaches. ALPHA should be a floating-point number between 0 and 1. For further information, see chapters 3.7 and 4.5.2.3.

XTLSHP This parameter holds (in floating-point format) the maximum permitted difference between the squares of the diameter of the crystal (in millimeters) in two adjacent sections of the crystal, approximately 1.2 millimeters apart from one another. The square of the diameter stored for buoyancy compensation purposes is adjusted, if necessary, to differ by not more than the value of XTLSHP from the preceding value.

Display Control:

INTRVL This Variable determines the duration of the intervals between subsequent output operations to the console. One unit corresponds to an interval of 50 milliseconds. The default value of 10 corresponds to a complete screen update every four to six seconds, depending on the other activities within the CGCS. More frequent updates may be required during testing and alignment; they can be achieved with smaller INTRVL values. The fastest screen update is done with INTRVL set to 1; a zero INTRVL value disables the screen output entirely. Note: The screen display will "freeze" irreversibly if INTRVL is set to zero; regular operation will not be resumed even if INTRVL is set back to a nonzero value. The system has to be restarted in order to re-activate data output on the screen. (The CGCS remains operable, though, with the screen output disabled.) INTRVL does not affect the output of the time, of operator commands, and of system messages.

Data Dump Control:

DUMPIN The Variable DUMPIN holds the interval between periodical Data Dumps to the Print file; the time units are minutes. DUMPIN may be set to any convenient value at any time; a DUMPIN value of zero disables the periodical Data Dumps.

DUMPFL This Variable triggers an additional Data Dump (and an additional record written to the Data file) if it is set to -1. Note that a SET DUMPFL -1 0 command is the only save way to trigger additional Data Dumps from a Macro Command file. (DUMPFL is reset by the Data Dump routine; it must therefore be set to -1 repeatedly if more than one Data Dumps are required.)

Scratchpad Variables:

| | |
|---|---|
| DUMMY | In order to facilitate advanced Macro programming, eight dummy INTEGER*2 locations were provided. These locations are not accessed by the CGCS code proper, but they may be arbitrarily ramped or used as flags (set to specific values) and employed in Conditional Macro commands. The dummy locations are referred to as DUMMY(1) through DUMMY(8). |

Miscellaneous - Read-Only Variables:

| | |
|---|---|
| TIME | The Variable TIME holds the current system time (in seconds) in an unsigned two-byte INTEGER location. This counter wraps around to zero after 65,536 seconds. Note that the contents of TIME are interpreted as a signed INTEGER*2 number by the display and also by the Conditional Macro Command execution routines; time counts greater than 32,767 seconds are thus interpreted as negative numbers. |
| RAMPNG | This Variable holds the number of parameters which are currently being ramped. You may look at it (and have your Macro commands look at it), but messing around with RAMPNG will inevitably confuse the CGCS. The results may be spectacular but probably not desirable. |
| CNDCNT | The same considerations as to RAMPNG apply to the count of pending Conditional Macro commands kept in this Variable. |
| ZERO | This location holds, simply enough, a zero INTEGER*2 value. You may try to modify it but you won't be very successful since this location is in ROM and thus inaccessible to any writing attempt. |

## 3.7 THE DIAMETER EVALUATION ROUTINE

The control operations of the CGCS center around the control of the shape of the crystal grown, i.e., the control of its diameter. Unlike the approach used in the conventional analog controllers for compound crystal growth, it is the diameter rather than the first derivative of the crystal weight which is compared to a pertinent setpoint and whose deviation from the setpoint is used as an error signal for a PID based controller algorithm (compare chapter 4.5.1.) The accuracy and usefulness of the diameter control approach depends therefore crucially on the accuracy of the calculated crystal diameter.

The diameter evaluation approach used in the CGCS is based upon the differential weight signal supplied by an analog differentiator circuit. After its A/D conversion, this signal is submitted to digital low-pass filtering; an anomaly compensation analogous to the approach used in the Cambridge Instruments Anomaly Shape Control board may be applied to it. The current diameter of the crystal is calculated from this differential weight using the actual growth rate (i.e. the difference between the seed and crucible lift speeds plus the speed with which the semiconductor melt drops when it is consumed by the crystallization process). A full compensation for the buoyancy in the boric oxide encapsulant is provided; the diameter evaluation routine keeps track of the shape of the part of the crystal next to the solidification interface (to be accurate, of the last 75 millimeters of the crystal), and calculates the volume immersed in the encapsulant and the height of the boric oxide layer from this information. This approach permits the use of actual physical parameters of the system (like densities, dimensions, and speeds) rather than the modified parameters required in conventional analog growth.

As a by-product of diameter evaluation, the same routines provide a setpoint for the position of the crucible which is used in Automatic mode (Mode 4) to control the crucible lift speed via a PID loop. This control loop tries to keep the surface of the semiconductor melt (and thus the solidification interface) at the same location within the heater's hot zone despite of the dropping of the melt when molten material is consumed by the growing crystal.

The diameter and crucible position evaluation algorithms which are used throughout the major part of a crystal growth run are based on the following assumptions:

(1) The crucible is a straight right cylinder.

(2) The amount of boric oxide encapsulant remains constant.

(3) The semiconductor melt fills the entire diameter of the crucible, and material added to the crystal reduces the height of the semiconductor melt in accordance with the conservation of the total mass (melt plus crystal).

While assumption (2) is reasonably justified in the case of gallium arsenide throughout the entire growth process because the boric oxide encapsulant does not wet the crystal, this does not apply to the other two assumptions towards the end of a growth cycle: The transition between the crucible wall and bottom is always a bevel with a finite radius; and the semiconductor melt tends to contract itself due to surface tension and recedes towards the center of the crucible if its amount drops below a certain limit. In an extreme case, the above assumptions have to be amended as follows:

(1) The semiconductor melt forms a cylindrical disk with constant thickness whose diameter (rather than thickness) decreases in order to supply the material being solidified in the crystal.

(2) The gap which opens up therefore between the semiconductor melt and the crucible wall is filled with boric oxide encapsulant, which reduces the effective boric oxide height as the crystal grows.

The diameter evaluation algorithms used in the CGCS are capable of handling both extreme cases, and any arbitrary intermediate stage, according to the value of the Variable ALPHA. An ALPHA value of 1 corresponds to the first set of conditions (when the semiconductor melt fills the entire crucible diameter), whereas a value of 0 conforms with the second set (i.e., extreme melt recession). Values for ALPHA between 0 and 1 permit to model an intermediate stage between the two extremes in a heuristic mode: Most likely, the disk formed by the receding melt <u>does</u> reduce its thickness when the melt is used up by the growing crystal; the speed with which it does so may, however, be considerably less than during the regular growth. An ALPHA value less than 1 but still greater than zero will therefore be appropriate during the final growth stages. Since crystal growth will always start under conditions corresponding to the first set of assumptions, ALPHA is initialized with 1, and remains at this value unless it is explicitly SET or CHANGEd to a different value.

The RESET command is closely linked to (and required by) the diameter evaluation routines. It initializes the shape information required for the buoyancy compensation under the assumption of a cylindrical seed with the diameter specified with the INITIALIZATION command (or sequence) which passes through the entire boric oxide encapsulant layer, and it provides initialization values for the crystal length and weight calculation. Furthermore, a RESET command resets ALPHA to 1 and cancels all effects of a possibly different previous ALPHA value.

# 4. THE CZOCHRALSKI GROWTH CONTROL SYSTEM SOFTWARE

## 4.1 PROGRAM STRUCTURE

From the programmer's point of view, the Czochralski Growth Control System (CGCS) is an iRMX-80 based real-time application system consisting of a number of RMX "tasks". A task is a section of program code, usually dedicated to one control commission or part of it. It is more or less independent from other tasks and is executed whenever its specific action is required and system resources are available, according to the priority level which has been assigned to it. The execution of a task is scheduled by the operating system's "Nucleus", either in response to extraneous events (interrupts), or when a task receives data in the form of a "message" from a fellow task which it was waiting for.

From the user's point of view, however, the CGCS consists essentially of three functional groups (Fig. 7) each of which, in turn, is constituted of several tasks:

(1) The System Interface: This part of the software is transparent to the user (and therefore not shown in Fig. 7). It provides, nevertheless, essential functions like data formatting or timekeeping.

(2) The Operator Interface: These tasks form the link between the operator and the controller routines proper. Holding the system's "intelligence", they constitute the by far largest part of the CGCS code. The Operator Interface is responsible for the following actions:

  (a) Prompting for and interpretation of operator commands which control the functions of the CGCS.

  (b) Execution of operator and Macro command file sourced commands. This function was kept strictly separate from the operator command interpretation in order to facilitate the handling of Macro commands.

  (c) Recording of all commands pertaining to the actual crystal growth process.

  (d) Periodic output of measured data on the console CRT terminal, and to a disk file, and preparation of data to be output on an analog chart recorder.

(3) The Process Controller proper: These routines are actually involved in controlling the heater power(s) and motor speeds according to the pertinent setpoints provided by

the Operator Interface. They also constitute the interface to the analog and digital I/O hardware.

We will follow the above scheme for the subsequent discussion of the CGCS software.

## 4.2 GENERAL PROGRAM INFORMATION

The CGCS consists of routines part of which were written in Fortran, part in assembly language. In general, the Operator Interface and part of the actual Controller routines are Fortran based, whereas the System Interface modules (and all system routines which were not supplied by Intel) consist of assembly language code. Assembly language was chosen when one or more of the following requirements had to be met:

* Interface to iRMX-80 system routines which cannot be called directly from Fortran due to different parameter passing conventions.

* High operation speed, which is particularly important if a routine is invoked very frequently.

* Numeric operations which can be coded more efficiently in assembly language than in Fortran (e.g., the low-pass filtering algorithm).

Fortran, on the other hand, was chosen where the use of a high-level language was considered advantageous with regard to program clarity and programming efficiency. It was the obvious choice for routines which involve floating-point arithmetics (because Fortran is the only compiler-based language supplied by Intel for 8080/85 processors which supports floating-point operations). In order to improve the execution speed and code efficiency of Fortran, a set of library routines was specially prepared which replace the standard (lengthy and slow) Fortran floating-point algorithms by routines which make use of the 8231 Numeric Processor. These routines are not only several kilobytes smaller than the standard ones, they also boost the execution speed by about one order of magnitude.

A special approach was necessary to fit the CGCS into the available memory of less than 54 KBytes. (More than 10 of the total 64 KBytes are required for the ROM resident system and its data areas in RAM.) The entire code of the Czochralski system would have exceeded this limit by far. We had, therefore, to choose an overlay approach (Fig. 8): Program code which is not required permanently within the system is loaded into a reserved memory area only when needed, overwriting another currently dispensable overlay. The only system function where this is possible without unduly impeding the system operation is the Command Interpreter which controls the dialog between the operator and the system. Since the operator can only enter one command at a time, and since human command entry is a very slow procedure, compared to the standards of a microcomputer, it was possible to split the Command Interpreter's functions into a total of 22 different overlays each of which is in charge of one particular command or a group of related commands. According to the size of the greatest overlay, a memory area of 2 KBytes was reserved for Command Interpreter overlays; the total size of all overlays together is approximately 30 KBytes.

The layout of the Czochralski Growth Control System memory map (Fig. 6) was chosen to permit an easy software updating. Modifying internal system parameters easily (compare chapters 1.3 and 3.6) requires a translation table which correlates the symbolic name of a system "Variable" to its physical storage location in memory. Since this translation table has to be generated manually, it is obviously not desirable if it has to be rewritten after each minor modification of the controller software. The system grows or shrinks at its high-address end; in order to prevent them from being affected by system size changes, all important system Variables were located at the lowest addresses available, immediately above the code and data areas of the ROM resident system. Most of this data must be available to several system tasks; extensive use was therefore made of named Fortran "COMMON" blocks which are arranged (in alphabetical order) at the lowest addresses and consume approximately 1,280 bytes. They are immediately followed by the general system data area. The lowest addresses within this area are used by the data locations of assembly language modules some of which have to be "tied" to "COMMON" blocks; these locations are still not very likely to be affected by program modifications. They are followed by the data areas of the permanently resident Fortran based software which are essentially scratchpad locations for the internal use of these routines. The remainder of the data area whose total size is approximately 9,900 bytes holds system data which hardly need be explicitly accessed and whose actual absolute addresses do, therefore, not matter.

A 2 KByte range immediately above the data area is reserved for the Command Interpreter overlays' code and local data. It is succeeded by the bulk of the system code. This code area has currently a size of about 39.4 KBytes; the memory area between its top and some disk buffers and system variables which reside close to the absolute top of memory is used as a memory pool from which memory can be dynamically assigned to system tasks when required. The size of this memory pool does not matter unless it becomes too small; the program code may therefore grow due to software improvements without penalty. (The memory reserves are currently in the order of 1.5 KBytes, which does permit program improvements but certainly not the introduction of major new features.)

## 4.3 THE SYSTEM INTERFACE

The System Interface consists of a number of subroutines (which can be called by any task) and of five primary and several secondary tasks which are dynamically generated by the primary tasks. Interface routines which belong to a special set of interface libraries can be recognized by their names which begin either with "FR..." or with "FX...". A complete documentation of this software is contained in the Fortran-RMX-80 Interface Manual (see Appendix A). We will restrict ourselves to the discussion of two features of the System Interface which are not completely covered by the above documentation, and which are actually noticeable to the user, namely, Console and Printer I/O, and Timing.

### 4.3.1 CONSOLE AND PRINTER I/O

The user-friendliness of a system depends to a large degree on the design of the command input and data output routines. Input on the system console should be possible in a straightforward way, without requiring the operator to care for internal peculiarities of the system. Usually, input actions are explicitly requested from the operator, and operator input is accepted _after_ it was requested by the system. In real-time systems, however, it may happen that the operator-system dialog is delayed due to an extraneous event which requires immediate response, and the operator may be ready to enter console input _before_ the system is waiting for it. In order to prevent operator entries from being lost or truncated, the system should have a "type-ahead" feature which collects in a special buffer operator entries which were not yet requested for. The specially written Terminal Handler used by RXISIS-II and the CGCS offers, in fact, very comfortable type-ahead; up to 80 characters of operator input which may be contained in a maximum of 40 input lines may be collected in the type-ahead buffer.

The design of the output to the operator's console is not straightforward either: The standard approach is chronologically arranged output where new items are added in the bottom output line, and older output lines scroll up and, on a CRT terminal, eventually off the screen. Although such a scrolled output is perfectly fine if it need only contain the dialog between the operator and the system, it is much less suitable if a large amount of data is to be displayed, particularly if output data is generated asynchronously by various processes. In this case, a CRT screen with a fixed format offers clearly advantages: each item can be found at the same place of the screen all the time, and there is always a complete set of valid output data displayed. Each item need only be written to the console when it was changed, which evidently saves a lot of output overhead.

However, the standard Fortran output routines (and also the standard iRMX-80 Terminal Handler) support only simple scrolled "Teletype" output. A special set of I/O routines and tasks was therefore prepared which provide, in conjunction with the improved Terminal Handler, the following features:

* Support of a "fixed" output screen, i.e., the possibility to write in a random mode to any location on the screen of a console CRT terminal.

* A "split" output screen which provides, in addition to fixed format output, a conventionally scrolled area to which data can be written whose chronological order does matter. This scrolled portion of the screen can advantageously hold echoes of the operator's input, and system messages.

* Various output data formatting functions which are not provided by standard Fortran, e.g., a self-adjusting floating-point number output format.

* Support of command line parsing techniques. An input line may be processed repeatedly and scanned for various items.

* User-friendly unformatted input of numeric values. In contrast to standard Fortran, hardly any syntax rules need be regarded.

In fact, all standard Fortran I/O routines, including disk I/O, were replaced by specially written alternative code. Although the programming interface of the alternative routines is a little awkward compared to the standard Fortran ones, the specially written modules comprise not only considerably less program code, they are also much faster than their Fortran counterparts.

Some output items, in particular, the entire dialog between the operator and the system, should also be recorded for documentation purposes either on the printer, or on a disk file. A special set of interface routines was therefore prepared which can be called by any task which requires input or generates output, namely, STRIN and DATIN for the input of data to variables of type CHARACTER and of any other type, respectively, and STROUT and DATOUT for the corresponding output operations. (Fortran passes variables of CHARACTER type to subroutines and functions in a way different from any other variable type, which necessitates a separate treatment.) The input routines echo the entire input line to the documentation output, while the output routines write simultaneously to the screen <u>and</u> to the documentation output. In either case, each documentation output line is preceded by the actual and the internal system time of its generation. The documentation routines format their output into pages of 56 lines each; each page is headed by a line which holds the run's date, a run identification, and a page number.

The peculiarities of a real-time process control system require an extremely high degree of fault tolerance, particularly for the I/O routines. The failure of a peripheral (and, possibly, only auxiliary) device like a printer must by no means permanently detain the operation of the remainder of the system. Therefore, a printer timeout feature was provided which discards printer output if the printer did not respond within a given period (currently, 10 seconds); after three unsuccessful attempts to write to the printer, printer output is disabled altogether. (A corresponding error message is displayed on the CRT console.)

### 4.3.2 SYSTEM TIMING

Time is not only a crucial factor in process control, a timing facility of sufficient accuracy is also very desirable for any data logging or operation recording. Unfortunately, there is no reasonable clock hardware available within the family of OEM boards chosen; timing has, therefore, to rely on software routines.

The heart of the timing of the CGCS is a dedicated timer task (FXTIME) which is triggered by the internal timing of the iRMX-80 operating system. iRMX-80 permits to introduce delays in the execution of a task which are multiples of 50 ms, its internal time unit, which is derived from the on-board quartz clock. FXTIME runs once every second, i.e., every 20 iRMX time units. It provides time information (in the format hh:mm:ss) of the actual time in 24 hours notation, and of an internal system time which starts when the CGCS is loaded, and which is counted up to 95 hours, 59 minutes, and 59 seconds, wrapping around to zero after 96 hours. Furthermore, FXTIME features two counters which are incremented every second and which wrap around to zero after 65536 seconds (approximately 18 hours and 12 minutes). One of the counters runs from the system start, the other can be reset arbitrarily, and used to trigger an "alarm clock" when its count exceeds a specified value. FXTIME also sets flag bytes once every second, every ten seconds, every minute, and at programmable intervals; these flag bytes and the "alarm clock" function are used to trigger the execution of controller tasks. In fact, all critical timing is derived from FXTIME, which guarantees an accurate long-term synchronization of tasks. The internal timing of the CGCS can easily be checked by comparing the displayed time to an accurate clock. (The timekeeping of the CGCS is not extremely accurate, though, compared to the standards of common quartz clocks. This is true because it is based on the quartz oscillator on the CPU board which was designed only to be sufficient for providing a microprocessor clock signal; it does certainly not meet the requirements of high-accuracy timing.)

## 4.4 THE OPERATOR INTERFACE

### 4.4.1 THE CONSOLE CRT SCREEN

The output on the CRT console terminal is the major visible part of the CGCS's Operator Interface. Several tasks some of which are not even part of the Operator Interface proper contribute to the console output (compare Fig. 9):

(1) Fixed Part (Lines 1 through 16 or 17):

Timer Task (FXTIME):
Actual and system time.

Command Interpreter Task (RXIROM):
Table frames, text output, date, and run identification.

Command Executor Task (CMMDEX):
Macro command name; operation mode.

Measured Data Output Task (MEASDO):
All numeric values; Debug output in line 17 if activated.

Command File Input Task (CMFINP):
Macro command name (cleared).

(2) Scrolled Part (Lines 17 or 18 through 21):

Command Interpreter Task (RXIROM):
Operator entry echoes, various messages.

Command Executor Task (CMMDEX):
Various messages.

Command File Input Task (CMFINP):
Various messages.

Diameter Controller Task (DIACNT):
Various messages.

All other tasks:
Disk, I/O, or system error messages.

(3) Prompt Line (Line 22):

Command Interpreter Task (RXIROM)

(4) Input Area (Lines 23 and 24):

Directly written to by the Terminal Handler.

The numeric values written to the console are, in general, given as physically relevant magnitudes, i.e., as properly scaled floating-point numbers. The following dimensions apply to the various items:

* Diameter, Lengths, Positions: Millimeters.
* Temperatures: Millivolts (thermocouple voltages).
* Lift Speeds: Millimeters per hour.
* Rotation Speeds: Revolutions per minute.
* Weights: Grams.
* Differential Weight: Grams per minute.
* Powers, Contact Device: Arbitrary units (0 ... 100).

* Gas Pressure: Pounds per square inch.

* Densities: Grams per cubic centimeter.

## 4.4.2 AUXILIARY I/O ROUTINES

The tasks which request console input or generate console output (compare chapter 4.4.1) use, in general, the Fortran-RMX-80 Interface I/O routines whose names start with "FR..." to write to the screen, or the routines discussed in chapter 4.3.1 if they also write to the documentation file. All these output routines require a screen position information which is passed in the first parameter of the subroutine call. Some locations on the screen are, however, very frequently written to, and it was advantageous to provide special routines for these output actions which have the screen position information implicitly built in. Calling any of these "shorthand" routines spares the programmer entering one parameter, and it abbreviates the actual program code. Similarly, some input actions like the checking for the input string "Y(es)" can expediently be handled by dedicated routines.

The following routines (and several others) are kept in the assembly language module AUXASM. With the exception of PRETTA, they may be called by any task performing output.

PROMPT: This routine writes the string which was passed to it as a parameter left-adjusted into the input prompt line (line 22).

MESSGE: The string passed as a parameter to MESSGE is written into the scrolled screen area.

ERRMSG: Similar to MESSGE, the ERRMSG routine writes to the scrolled screen area, appending a "beep" in order to attract the operator's attention.

PRETTA: This routine writes "- press "RETURN" key to continue" to a specifiable screen location (usually in the prompt line), and waits for any input on the console.

Three additional I/O routines are kept in the Fortran module AUXCOM:

BEEP: This routine simply issues a "beep" on the system console. It takes no parameters.

CLIPRL: The subroutine CLIPRL overwrites the input prompt line with spaces. It does not take any parameters.

CHKANS: This routine is a LOGICAL Function. It returns ".TRUE." if a valid input line beginning with an upper- or lowercase "Y" was entered on the console, and otherwise ".FALSE.". CHKANS needs a LOGICAL argument which is returned ".TRUE." if an empty line ("Return" only) was entered, and otherwise ".FALSE.".

## 4.4.3 THE COMMAND INTERPRETER - TASK RXIROM

The Command Interpreter task has a special position among the CGCS tasks in several regards:

* It is, in fact, the continuation of the ROM resident part of RXISIS-II, RXIROM, and the first task to come "alive" in the CGCS. Although it is "unofficially" referred to as "COMINT" within the program source modules, we will use here its "official" name RXIROM (which is also reported, e.g., by disk error messages).

* It performs the system initialization and activates all other CGCS tasks.

* It is the only task which requests and processes operator input (but not the only task to generate output).

The Czochralski Growth Control System is invoked under RXISIS-II by the command "CZOCHR". RXISIS-II searches for and loads a program module "CZOCHR.RXI" whose only purpose is to vector control to a special code sequence in the RXISIS-II Command Line Interpreter which replaces the file name extension ".RXI" by ".BIN", provides the resulting module name "CZOCHR.BIN" for the ROM resident bootstrap routine, and restarts the system. The bootstrap routine is part of the task RXIROM; normally, it loads into RAM and starts RXISIS-II. Being entered in the described way, however, it loads the module "CZOCHR.BIN" rather than "RXISIS.BIN" from disk drive 0; "CZOCHR.BIN" holds the entire resident code of the CGCS plus preliminary initialization values for some data locations, and a special start module which is loaded into the memory area which will later be used by the Command Interpreter overlays. Control is passed to this initialization code when the program file was successfully loaded.

The start module is entered via the assembly language routine CZINIT which first sets an internal flag of the Monitor which enforces a duplication of the Monitor's CRT output to the printer. (This measure provides a permanent record on the printout of an inadvertent entry into the Monitor program which might happen due to software or hardware failures.) CZINIT also resets a flag which controls the activation of the Monitor from the console keyboard. (This is why the Monitor can be entered under RXISIS-II but not from the CGCS by pressing the "Break" key of the console terminal.) Subsequently, CZINIT builds a new task stack close to the top of memory since the stack of RXIROM is too small. (Each iRMX-80 task requires a stack of its own.) It stores a program version code in a reserved memory location; later, a version code which is loaded with each overlay will be compared to this data in order to ascertain that only matching program modules are loaded. After some initialization calls to Fortran and Fortran-iRMX Interface routines, CZINIT passes control to the Fortran subroutine FXUSIN.

FXUSIN initializes the digital I/O interface and several control structures which can be accessed more conveniently via Fortran than via assembly language. It calls the (assembly language) subroutine TESTHD which checks whether an A/D converter board is installed in the system by initiating a conversion and checking the status byte returned by the A/D converter for a "Conversion Ready" bit. The Variable TEST is set to -1 if no A/D converter response was detected within a defined timeout period; otherwise, TEST is returned with the value 0. (This check is important if the CGCS software is intended to be run on hardware which does not feature the A/D and D/A interfaces. In this case, practically all system resources would be spent by the task ANACNT for waiting for the A/D converter to finish a conversion, which obviously never happens if there is no A/D converter within the system. The CGCS would, therefore, be practically locked in such a test environment. The value of TEST is later used for bypassing the analog input and output routines within the task ANACNT. Note that the TESTHD call is done <u>before</u> ANACNT is created.) Subsequently, FXUSIN calls the assembly language subroutine CREATE which is, similar to TESTHD, part of the module CZINIT. CREATE activates all tasks of the CGCS, which can only be done safely <u>after</u> the above initializations, and makes unused memory (including the old RXIROM stack) available to the memory pool of the iRMX-80 Free Space Manager. After the return from CREATE, FXUSIN provides a sign-on message (plus a message referring to a "Test Mode" if TEST has been set to -1), and loads the data overlay "CZOOVD" from drive 0.

Similar to "CZOCHR.BIN", "CZOOVD" is loaded only once during every growth run. "CZOOVD" which holds the (initialization) values of practically all system parameters is kept separate from the main code module on purpose. The preparation of the CGCS program modules is a lengthy and complicated procedure which would have been indispensable after each modification of a system parameter initialization value if this data had been kept within "CZOCHR.BIN". Since it is very likely that numeric parameters require changes more frequently than the program code, it was preferable to load them from a special data overlay which can be modified and configured very easily.

The auxiliary routine LOVLAY which is exclusively used by RXIROM loads overlay modules into RAM. (The information where data is to be loaded is part of the overlay program file. It is, therefore, sufficient to specify the name of the file to be loaded.) Several safeguards are provided which permit to trap the potentially disastrous loading of improper files:

* The data on each disk file and, in addition, the program code itself, contains checksums which are validated by the Loader task. Any damage to a program file is therefore very likely to be detected and reported by the Loader. LOVLAY returns a message "Defective program disk" in this case.

* Each overlay contains memory locations which hold its name and the program version code. LOVLAY reports "Software damage likely - reset the system" if either the overlay name or the program version loaded with the overlay do not match the expected data. (It is important not to mix modules belonging to different CGCS versions because all overlays access code or data within the resident part of the CGCS. Since the absolute address of a routine or a data location may change due to system modifications, an overlay routine may call improper code or access wrong data if its version does not correspond to the version of the resident code.)

Note: Do not disregard error messages returned during overlay loading. A potentially disastrous effect of a defective overlay may show up only after a considerable time. It is always dangerous to copy single overlay files to a work disk, or to exchange work disks inconsiderately. There is, however, no danger if a Disk Error 24 is reported during overlay loading, and if the defective disk is replaced by one which holds the same program version.

(In very rare cases, a Disk Error 120 - Unable to Open File - may be displayed when the system attempts to load CZOOVD. This may happen if the operating system is overburdened during the start phase, for example, if a key is continuously pressed on the console terminal. In this case, the memory pool has not yet been initialized when memory is requested from it by the Loader software, and the above error condition ensues. The "Defective program disk" message may be ignored in this case, and loading may be retried by pressing "Return".)

The start routine FXUSIN displays the creation date of the data overlay CZOOVD (which is also an indication that this module was loaded properly), and requests the current date. The date should be entered in the format shown in the prompt, but any string of 8 characters which starts with a digit is accepted. The date information is stored for reference purposes only; it will be used on the console screen, in the documentation output page headers, and in the header records of the Data files. After the date, the current time is requested from the operator; the system expects two or three positive integer values as an input, separated by colons (":"), spaces, or any other non-numeric characters. The time should be entered in 24 hours format; zero is assumed as a seconds value if only hours and minutes were specified. The internal system time starts running - yet invisibly - when the subroutine FRSETT is called after the "Return" key was pressed to enter the time information, and the absolute time is set to the value entered. Finally, FXUSIN requests a "Run Identification" which can be any arbitrary string up to 20 characters long. A blank run ID can be entered by simply pressing "Return".

FXUSIN calls now the subroutine TIMLIN which is part of the start code in the future overlay area. TIMLIN generates the date, absolute time, run ID, and system time display in the top screen line which will be shown throughout the entire growth run. The operator can accept or reject the data displayed; this is, by the way, the only occasion within the entire CGCS where a plain "Return" is interpreted as "Yes" (otherwise, it is treated like "No"). Depending on the outcome of this query, FXUSIN either loops again through the date, time, and run ID input section, or it returns to CZINIT which passes control to the resident portion of the Command Interpreter, i.e., to the routine COMINT.

COMINT starts its operation by writing the output "frame" to the console terminal which is eventually filled in with the output of measured data. This is done by the subroutine FRAME which resides in the overlay CZOV08. This overlay has to be loaded by COMINT; it overwrites the code of CZINIT and FXUSIN. (This and the following initializations can, therefore, not be done from FXUSIN which would otherwise be the logical place to do them; there is no way for a routine in an overlay to call directly another overlay resident routine which uses the same physical memory locations.) We will discuss the subroutine FRAME later which is also called upon a RESTORE command.

The next two routines invoked during the initialization of COMINT reside in overlays CZOV16 and CZOV19, respectively. DOCUMT permits to activate a Documentation output, either on the printer or on a disk file, and it allows to specify an interval for dumps of measured data to the Documentation output. INIDAT permits the initialization of some process parameters. Both routines will be dealt with later.

COMINT enters now its infinite loop which starts with the output of the prompt "Please command:" and the request of operator input. The input routines transfer an input line of up to 80 characters into an internal buffer when the operator terminates his entry with "Return"; no data is available to the CGCS before "Return" is pressed. First, COMINT attempts to transfer the first six characters in this input buffer to the CHARACTER variable COMMD. The LOGICAL variable STAT is returned ".TRUE." by the STRIN call if and only if an empty line was entered ("Return" only). COMINT repeats its input prompt in this case, and waits for the next entry. Otherwise, the first character of the input line is checked and the input rejected if it is a space (the command keywords must be left adjusted within the input line to be processed properly).

The presumable command keyword in COMMD is now compared to (currently) 25 keyword strings, corresponding to the 24 Internal commands (the HELP command has the alternate keyword "?"). Control is vectored to the appropriate sequence within COMINT if a matching string is found. (The string comparison routine FRCMPS uses the character "|" as a wild card symbol which can be matched to any character; only four characters are compared since the keyword strings consist of four or less characters only.) The command entry is interpreted as the name of a Macro command if no matching Internal command was detected. A valid Macro file name string is created by the assembly language routine MAKEFN which appends the file name extension ".CMD" at the logical end of the presumptive Macro command name (which is either after the sixth character of COMMD, or at the first space in COMMD, whatever happens first). COMINT tries to open the Macro file for reading, and closes it immediately, in order to test whether a file with the specified name and the extension ".CMD" does exist. This is the case if no error status is returned by the FROPEN call; FROPEN will return an error value of 13 ("No such file") if no Macro file was found with the specified name, most likely due to a mistyped command. An error value of 4 ("Illegal file name") may be returned if the command input contains non-alphanumeric characters, which may also happen due to typing errors.

COMINT returns to the beginning of its command loop with an appropriate message in these cases; the standard disk error message is output if any other disk error was detected. If a file with the proper name was found, COMINT assumes that it is a valid Macro file (this fact will be checked later); it requests an operator acknowledgement ("Execute Macro command ...?"), and sends a command message to the Command Executor which eventually will start the execution of the Macro command.

In general, all commands which may be recorded on and issued from a Macro command file are executed by the Command Executor. These commands are "sent" to the Command Executor by means of messages, buffer areas in RAM which are made available to the receiving task by the iRMX-80 operating system. Command messages have a "type" value of 161 (the message "type" is simply a safety feature which guarantees that correct data is received). The first byte of the command message proper determines the command type (in our case, 30H stands for "Macro Command"), and the remainder of the message holds parameters of the specific command, up to a length of 13 bytes. The same format, with two additional leader bytes holding the command time, is used to store commands within a Macro file; compare Appendix H. Using special message transmission routines of the Fortran-RMX-80 Interface Program Package permits to easily merge command messages from different sources (namely, from the Command Interpreter and the Command File Input tasks) and to queue them at the Command Executor's input for processing.

Most of the Internal commands are processed in overlay resident routines which we will discuss later, rather than within the main Command Interpreter routine COMINT. This approach helped to keep the resident COMINT code concise. Only the following commands do not require overlays to be loaded:

EXCHANGE: This is considered an emergency routine which must be called if a disk has to be changed due to any kind of defect. It would not make sense to load an overlay from a possibly defective disk. In order to process the EXCHANGE command, COMINT calls the Fortran subroutine XCHDSK which closes all files on the specified disk, waits for an operator entry which indicates that the disk has been exchanged, and re-opens all output files on the new disk.

END: An End of Command Record code (7FH) is sent to the Command Executor if this command was issued.

QUIT: The Fortran subroutine QUITCM which is invoked by COMINT disables the Macro command file input (by resetting the proper I/O flag) and the Timer #2 which controls the execution of Macro commands. It fakes a timer alarm by setting the flag TIMINT, and waits for two iRMX-80 time units (100 ms) to permit the Command File Input task CMFINP to run in response to the faked alarm. CMFINP closes the Macro command file, clears the Macro name on the top line of the console screen, and issues a corresponding message if it finds the I/O flag reset.

DUMP: The subroutine DUMP which is called immediately upon a DUMP command sets a flag (DUMPFL) to .TRUE. whose status is periodically checked by the Command Executor (compare chapter 4.4.4.6). The Command Executor, in turn, initiates a Data Dump to the Documentation output when it finds this flag set.

All other commands are handled by the overlay resident routines. In order to avoid loading an overlay which has already been loaded by a preceding command, COMINT checks the value of the variable OVRLAY which is set by each overlay to its respective overlay number. (This is not explicitly done by program code but by assigning a value to OVRLAY with a BLOCK-DATA program; this value is stored in OVRLAY when the overlay is loaded.) The COMINT overlays are discussed in the following chapters in their numerical order which has been determined essentially by historical reasons.

#### <u>4.4.3.1 OVERLAY CZOV01 - MODULE SETPAR - COMMANDS SET AND CHANGE</u>

The subroutine SETPAR receives the MODE switch as a parameter which distinguishes between the SET and CHANGE commands, and it returns the LOGICAL variable LOAD. LOAD is returned ".FALSE." if SETPAR can complete the processing of the command, i.e., if the command applies to one of the nine Internal parameters (diameter, three temperatures, four motor speeds, and power limit). Otherwise, LOAD is ".TRUE.", and COMINT has to load the overlay CZOV02 in order to complete command processing.

SETPAR re-scans the command line originally issued to COMINT, searches in it for the first space, and then for the first three alphabetic characters after the space, in order to determine the parameter which is to be SET or CHANGEd. An explicit request for a parameter is issued if no data is found in the input line, and a new input line is read and parsed for its first three characters. The command is cancelled if this second attempt is also unsuccessful. In either case, the input line pointer is moved back to the beginning of the parameter string, i.e., the next input command will read the parameter string again unless a search option is used with the input routine call. The three input characters are now compared to the nine mnemonics which stand for the primary parameters (three characters are required because the third of them must be a space in order to match a valid mnemonic). SETPAR is left immediately, with LOAD set ".TRUE.", if no matching mnemonic is found.

The routine scans now to the first space after the parameter string and tries to read a valid floating-point number from the input buffer. This number will represent the target value of a SET command or the increment of a CHANGE command. A proper value is requested if no numeric value is found after the parameter string, and a new input line is read in this case. Either input line is scanned for the next (floating-point) number, and a transition time entry is prompted for if no such number or a negative value is found. The command is regarded cancelled if no valid input is entered after it was explicitly requested. A similar approach is used within all Command Interpreter routines which process commands which permit the entry of command parameters in the input line.

In order to generate an operator confirmation prompt, SETPAR determines now the final value of the modified parameter. This value is equal to the input value for a SET command but must be calculated as the sum of the current parameter value and the specified increment in the case of a CHANGE command. Internally, the setpoint and actual values of the primary parameters are stored as scaled two-byte integer (INTEGER*2) values. This was done because analog data is input and output as integer values; the controller routines operate on integers because integer algorithms are faster and require less code, and data recorded in the Data file is also in integer format, which reduces the Data file size by a factor of two, compared to floating-point numbers. The physically relevant (floating-point) data which is displayed and entered on the console is obtained from the internal integer values by multiplying them with appropriate scaling factors.

One peculiar property of real-time systems must be considered at this point: Unlike conventional computer programs, routines which are part of a real-time system may not freely read and write data. This is true because multi-byte values are usually stored and retrieved in sequences of several machine-code instructions. The scheduling of system tasks is, however, hardly predictable in a real-time environment, and a task might be interrupted, e.g., during a multi-byte read, by another task which might write to the same memory locations.

Although the actual value stored in these memory locations might change only slightly, a totally unusable value might be retrieved by the interrupted task. Such an event may be relatively unlikely but nevertheless disastrous; the following safety measures are taken within the CGCS to prevent it:

(1) Some data areas are protected by access control routines (FRACCS and FRRELS) which permit only one task at a time to read them or write to them.

(2) The system Variables are implicitly protected by the proper choice of the priorities of tasks which access them. They are only written to by the Command Executor which has a very low priority and can therefore never interrupt the execution of a higher-priority task which might use a Variable. The storage of the Variables is protected by using a special routine (STODAT) which temporarily disables the system interrupts.

(3) Values which have to be read only can be retrieved reliably by reading them twice. This process can be repeated until both reads result in the same value.

The latter approach is the one chosen in SETPAR; a counter prevents the system from being blocked in the unlikely case that a matching value pair is never found.

SETPAR checks the final setpoint for negative temperature or power limit values, and requests an operator acknowledgement. The output line has, unfortunately, to be built relatively awkwardly in a buffer (LINBUF): The output routines which write also to the Documentation file can only accept a complete line of output.

Upon a positive answer of the operator, SETPAR builds the command message. The command type byte holds the encoded command mode (SET or CHANGE) and the target parameter; the input value is converted to an INTEGER*2 (which is checked for a potential overflow), and the transition time value which was specified in minutes is multiplied by 60 to hold a ramping time in seconds. The command message is dispatched to the Command Executor, and SETPAR returns to the resident COMINT code.

## 4.4.3.2 OVERLAY CZOV02 - MODULE SETVAR - COMMANDS SET AND CHANGE

SETVAR is invoked after a SET or CHANGE command for which none of the Internal parameters was specified. The CGCS assumes in this case that the command applies to a system Variable, i.e., to an item in a list of named memory locations. SETVAR receives the input buffer from SETPAR with the pointer at the first character of the presumptive Variable name; it reads a string of up to 10 characters into an internal buffer, terminating the input action when a space (i.e., the end of the Variable name) is encountered. The name string is converted to uppercase, and passed to the assembly language routine FINDAD.

FINDAD compares the presumptive Variable name in VARNAM to a list of names kept in the specially formatted file CZONAM.Vmn, where m and n are the major and minor program version numbers, respectively. Each entry in this file holds a Variable name (1 to 6 alphanumeric characters long, but the first character must be alphabetic), the Variable type (one- and two-byte integers or four-byte floating-point numbers), encoded with the number of elements if the Variable name refers to an array, and the Variable address or the start address of an array (compare Appendix H). FINDAD checks the index of an array element which may optionally be passed in parentheses immediately after the Variable name, and returns the actual address of the Variable or array element, and a type code which is positive if a valid entry was found in the CZONAM file, and negative in case of any error.

SETVAR checks the type code returned and issues an error message if necessary; otherwise, it retrieves the current value of the Variable. This is done with a call to the assembly language subroutine PEEKDW which reads the four bytes at the address passed as a parameter repeatedly until a stable result is obtained (compare chapter 4.4.3.1). The four bytes read may have to be converted to a floating-point number according to the type of the Variable; the result of this operation is later used to display the current and the final values of the Variable. Subsequently, the routine tries to obtain a SET or CHANGE final value and a transition time from the input buffer, and it issues corresponding prompts if no data is found.

Similar to SETPAR, SETVAR checks integer values for a valid range, builds a command message if the operator acknowledgement was positive, dispatches the message, and returns to COMINT.

### 4.4.3.3 OVERLAY CZOV03 - MODULE COMMEN - COMMAND COMMENT

The routine COMMEN inserts a comment line into the Data file if such a file is active.

COMMEN scans to the first space in the original command input line, and tries to read valid input from the remainder of the command line (to receive any comment which was entered together with the COMMENT command). A corresponding prompt is issued if the command line did not contain any data except the keyword. COMMEN returns immediately to COMINT if no Data file is active (i.e., IOFLAG(2) is reset); otherwise, it provides operation mode, time, and length grown information in its output buffer, sets the first byte of this 128 byte buffer to -1 to indicate a comment line, and writes the buffer to the Data file. It is essential that a full record (128 bytes) is appended to the Data file to maintain the file's special format (compare Appendix H).

### 4.4.3.4 OVERLAY CZOV04 - MODULES MENOUT AND CLRSCR - COMMAND HELP

This overlay provides the Help menus of the CGCS in response to the commands HELP and "?". It writes in random access mode into lines 17 through 21 which are otherwise reserved for scrolled and Debug output. The latter is immediately disabled by resetting the flag ENDBGO when MENOUT is entered. Although the output routines do permit to write over the scrolled area in random access mode, this output remains on the screen only until data is output in scrolled mode. Any system message which is issued while the HELP command is executed will therefore preempt the display of the current Help menu.

MENOUT first clears the five lines of the scrolled area by overwriting them with spaces (in the subroutine CLRSCR), and outputs a quick menu of Internal commands which is built right into the program. The next help screen optionally displayed by MENOUT contains a list of Macro command names which are derived from the disk directory of the disk in drive 0 (file ISIS.DIR). The directory is scanned for all valid files with an extension ".CMD". Up to 40 Macro files can be listed on one screen; if there are more Macro files on the system disk, MENOUT pauses and continues its output when the operator pressed the Return key.

After displaying the Macro commands, MENOUT permits to request more information about the Internal commands. If the operator accepts this offer, MENOUT displays again the short menu.

(The initial menu display sequence is also used for this purpose; a LOGICAL variable controls the continuation of the execution of MENOUT after the menu was output. This approach was chosen rather than a subroutine call because it is more program code efficient, and because it does not require awkward measures like COMMON blocks or lengthy subroutine parameter lists to make variables available to all routines involved.) Simultaneously, MENOUT opens the help file CZOMEN for reading which holds five lines of text for each command. There are two modes in which the contents of CZOMEN can be displayed: One mode steps through the file, displaying record by record, while the other one scans the file until a keyword entered by the operator is found in the first line of an entry; only this entry is displayed. Both modes can be combined since an empty input line ("Return" only) always results in the next record being displayed, whereas the first four characters of a non-empty input line are used to search through the file CZOMEN. Multiple entries can therefore be searched for in one pass, provided they are in ascending alphabetical order. A single-character entry (nominally, "Q", but any other character has the same effect) terminates the search, and MENOUT is exited after closing the menu file and re-enabling a possible Debug output by setting the flag ENDBGO.

#### 4.4.3.5 OVERLAY CZOV05 - MODULES OPMODE AND CLRSCR - COMMAND MODE

The operation mode setting routine OPMODE displays a mode menu similarly to MENOUT, and permits the entry of a mode number. The number entered is compared to the current mode and checked for its validity; corresponding messages are output if either the current mode was chosen, or if an illegal mode number was entered. OPMODE permits to re-select the current mode; although this has no effect whatsoever on the current growth run, the command is recorded in the Command Output file and may be effective during a later execution of this file as a Macro command. (It may also be used to trigger a data dump on the printer and in the Data file; there are more straightforward methods to achieve this, though.)

The operator is prompted for an acknowledgement of his mode entry in any case. OPMODE requests an extra acknowledgement (with "OK" rather than "Y(es)") if the mode is changed from Monitoring (mode 0) to any controlled mode, or vice versa, in order to prevent the probably disastrous effects which an inadvertent change might have. The newly entered mode is encoded in a command number, and the command message is sent to the Command Executor.

## 4.4.3.6 OVERLAY CZOV06 - MODULE DEBUG0 - DEBUG COMMANDS

The six DEBUG sub-commands - Continuously, Display, Modify, Off, Resume, and Suspend - are handled by the two overlays CZOV06 and CZOV07 (modules DEBUG0 and DEBUG1, respectively) which are concatenated similar to the two overlays for the SET and CHANGE commands. The command execution is commenced in the module DEBUG0 where the command input line is first scanned for the DEBUG mode switch, which is any one of the letters C, D, M, O, R, and S. As usual, a mode switch is requested if none or only an illegal one was found.

The processing of the DEBUG commands requires various interpretations of the input line, depending on which sub-command was issued. In order to facilitate this processing, the entire contents of the input buffer are read into an internal buffer (LINBUF) from which input items are retrieved. The contents of this buffer are shifted to the left by one item after each successful input operation, which permits to read the next item always from the beginning of the buffer. (Items must be separated by spaces; the buffer shifting subroutine SHIFTB simply advances to the first non-blank character after the first space and copies the buffer onto itself from this location on.)

For all sub-commands except Off, either the name of a Variable or an address is required as the first parameter. An input item starting with a number is considered a (hexadecimal) address, otherwise, the parameter is submitted to the routine FINDAD which was already discussed in chapter 4.4.3.2. The DEBUG routines distinguish between address and Variable input by setting the Variable type location VARTYP to -1 in the case of address specification, whereas values from 0 to 3 are returned by FINDAD for Variables.

Indeed, the information otherwise provided by FINDAD in the Variable type location must be obtained from the operator if address input was chosen since DEBUG would not know how to interpret the data at the specified address. (This information is not needed for the Display sub-command which outputs data anyhow in all perceivable notations.) A data format is, therefore, retrieved from the input buffer or requested from the operator if an address value was specified with a Continuously or Modify sub-command. (The formats used for numeric Variables are internally set to "I1", "I2", and "R", depending on VARTYP.)

The Continuously and Off sub-commands require the Debug Channel number, i.e., the number of the output location in the Debug line (1 to 4) which the command refers to. For both sub-commands, all necessary information is now available, and the proper command messages can be sent to the Command Executor.

The Display and Modify sub-commands display the current contents of the specified memory locations; in order to obtain this data, four bytes beginning with the given address are copied into local memory in an approach similar to the one used in SETVAR (compare chapter 4.4.3.2). This data is immediately displayed in several modes if the Display sub-command was issued: The four bytes or part of them are interpreted as ASCII string data, as an INTEGER*1 and INTEGER*2 variable, as floating-point data (type REAL), and as hexadecimal numbers. (A special treatment is necessary for the ASCII interpretation in order to avoid problems with data bytes which might correspond to control codes. Such bytes are replaced by periods (".").)

While the Continuously, Display, and Off sub-commands already have been completely processed when the end of the module DEBUG0 is reached, this is not true for the Modify, Resume, and Suspend commands. They have to be passed on to the second part of the DEBUG routine, DEBUG1 in CZOV07.

### 4.4.3.7 OVERLAY CZOV07 - MODULE DEBUG1 - DEBUG COMMANDS

Similar to SETVAR, DEBUG1 is only loaded if DEBUG0 returns with a status flag set to ".TRUE.". Data is passed between both routines by means of a special named COMMON block (DBGCOM) which is located at the top of the overlay area where it is preserved when DEBUG1 is loaded.

In order to conclude the processing of the Modify sub-command, DEBUG1 displays the current contents of the specified memory locations, and requests explicitly new data. Both values are displayed again for operator confirmation, and built into the command message if the confirmation was given.

The sub-commands Resume and Suspend which permit to resume and suspend the execution of an arbitrary task are treated essentially in common: Both require the address or the name of an iRMX-80 Task Descriptor as a parameter. Task descriptors which are referred to as Variables have the Variable type value of zero returned by FINDAD. Since specifying an address with a Resume or Suspend system call which is <u>not</u> the address of an iRMX-80 Task Descriptor would have a disastrous effect on the total system, multiple safeguards are used besides checking the VARTYP value: The name of the task, six alphanumeric characters, is stored in memory locations whose start address can be derived from the presumptive Task Descriptor. The command is cancelled if either non-alphanumeric characters are detected in the name area, or if the first character is not alphabetic. After an operator acknowledgement, a proper command message is again dispatched to the Command Executor which will, in turn, resume or suspend the specified task.

## 4.4.3.8 OVERLAY CZOV08 - MODULES FRAME AND TIMLIN - COMMAND RESTORE

This overlay provides the mask for the "fixed" output on the console CRT screen. It is executed upon a RESTORE command, and during the system initialization.

FRAME which is in charge of the main output mask first disables the output of measured data by resetting the flag RESTDO(3). This is important to avoid interferences between the two groups of output operations. Furthermore, Debug output in line 17 is suspended by resetting the flag ENDBGO. FRAME clears the CRT screen, and calls TIMLIN which restores the top (time) line. Subsequently, all fixed output items are written one by one, followed by a five line parameter dimension information written over the scrolled screen area. Having provided this menu, FRAME enables the output of measured data by setting RESTDO(3), and actually enforces data output by setting the remaining two flags of the array RESTDO. FRAME pauses then until the operator presses the "Return" key to permit him to read the display in the scrolled area. Writing a blank line into the actual scrolled output restores the previous contents of the scrolled area after a RESTORE command.

## 4.4.3.9 OVERLAY CZOV09 - MODULE FILES - COMMAND FILES

The subroutine FILES permits to display the current status of the three output disk files (the Documentation, Data, and Command file), and to change the status of a selected file.

First, FILES displays the name and the status of each file. The file names are kept in the CHARACTER array FILNAM; the file status is determined by the values of IOFLAG and FILLOC. The proper element of the array IOFLAG is set to ".TRUE." whenever a file is actually active, i.e., data can be written to it. FILLOC, in contrast, represents the physical location of a file; 0 and 1 stand for drive 0 and 1, respectively, and

2, for output to the printer. FILLOC is set to 3 if no file is open at all. In the case of the Control Output file, the setting of the flag RECORD has also to be taken into account which is ".TRUE." while commands are actually recorded (i.e., after a START command), and ".FALSE." otherwise.

The operator may now specify one of the three output files which he wants to be opened and closed, or return immediately to COMINT. The actual file treatment is performed by one of three separate overlays; the proper overlay number is determined by FILES and passed to COMINT in OVRLAY; COMINT concatenates the proper routine.

## 4.4.3.10 OVERLAY CZOV10 - MODULE REQCMF - COMMANDS START AND FILES

The subroutine REQCMF can only be called via the START and FILES commands; it opens, initializes, and closes Control (or "Command") Output files.

The response of REQCMF depends on the status of the file; it distinguishes between three cases:

(1) No Command Output file is open (IOFLAG(3) is ".FALSE.", and FILLOC(3) is 3).

(2) The file has been opened, but it cannot be written to due to a preceding disk error (IOFLAG(3) is ".FALSE." but FILLOC(3) is not equal to 3).

(3) The file is open and active (IOFLAG(3) is ".TRUE.").

In the first case, REQCMF offers the operator to open a Control Output file, and requests a file name if he agrees. A complete Macro file name is built from the operator's entry by appending ".CMD" (with the subroutine MAKEFN), and the resulting file name is checked for validity and for the drive where the file will be located (with CHKFNM). In order to prevent the accidental overwriting of ar existing file (if the operator entered the name of a file which already exists on the same disk), REQCMF tries to open the file with the specified name for reading first, and issues a warning if this procedure was successful, i.e., if a matching file was found. Otherwise, the Command Output file is opened for writing, and a header record is written to it. The header record holds zeros in its first two bytes (which otherwise contain the execution time of a command), and the system version code in the third and fourth byte. The remaining 12 of the 16 bytes of the header record are currently undefined. REQCMF finally sets IOFLAG and returns to COMINT.

In the second case, REQCMF permits either to re-activate the file (possibly, the error condition has already been corrected which set it inactive), or to close it. An open and active file may be closed only; if the operator agrees to close the file, IOFLAG and RECORD are reset, FILLOC is set to 3, and the file name string is deleted.

## 4.4.3.11 OVERLAY CZOV11 - MODULE CALCUL - COMMAND CALCULATE

The CALCULATE command constitutes a helpful utility which is, in fact, not connected to the crystal growth process at all. CALCUL permits to evaluate the sum, the difference, the product, and the quotient of two numbers. With regard to the requirements of the DEBUG commands, three formats are selectable for input and output data, namely, (two byte) Integer, Hexadecimal, and Real (floating-point). One set of instructions applies to the processing of floating-point input values, and an other, to integer and hexadecimal data. The results are displayed in decimal and hexadecimal notation in either case; the CALCULATE command may therefore be used to determine the internal (hexadecimal) representation of an arbitrary integer or floating-point value.

## 4.4.3.12 OVERLAY CZOV12 - MODULE DATAFI - COMMANDS FILES AND DATA

With the exception of the header record generation, the routine DATAFI which is responsible for the initialization and maintenance of the Data file is analogously identical to REQCMF (compare chapter 4.4.3.10).

A data sampling interval (in seconds) is requested from the operator when a Data file is opened; any value between 1 and 255 is accepted. The header record is built after the operator acknowledged the interval value. It contains the date, the run ID, the data records interval, and the system version. This header which is 32 bytes long is written to the newly opened Data file four times, to permit the first Data record to start at a disk sector boundary. (This is important because disk operations are much faster if an entire disk sector can be written or read.)

### 4.4.3.13 OVERLAY CZOV13 - MODULE EXICZO - COMMAND EXIT

This module has the chore of "closing down" the CGCS and the puller. It requires a double acknowledgement by the operator to be actually executed, in order to prevent accidental exiting from the CGCS. It performs the following operations:

(1) EXICZO disables periodic data dumps to the Documentation output.

(2) It sends an END command to the Command Output file if such a file is still open.

(3) It clears potentially pending Conditional Commands by transmitting a CLEAR command code to the Command Executor.

(4) It performs a QUIT command (calling QUITCM) to preempt a currently active Macro.

(5) It switches off Data recording by de-activating a Data file (setting IOFLAG(2) to ".FALSE.").

(6) It shuts the system down with the following actions if the digital system is actually controlling the puller:

  (a) It terminates automatic growth, changing the operation mode to "Manual" by sending an appropriate command message.

  (b) It terminates any parameter ramping possibly still in progress by resetting the Ramping flags RMPFLG (compare chapter 4.4.4).

  (c) It checks the current values of the motor speed and power limit setpoints, and enters into the following actions if any one of them is not equal to zero:

    (1) It permits the operator to skip from EXICZO and to shut the system down on his own account.

    (2) It ramps the power limit setpoint to zero within approximately 6 hours unless it is already zero, generating an appropriate command message.

    (3) It ramps the seed and crucible lift speeds to zero within one minute.

    (4) It provides a time countdown in the input prompt line which starts at 360 minutes if the power

limit need be ramped down, and otherwise at one minute.

(5) It ramps the seed and crucible rotation speeds to zero within one minute when the countdown display shows one minute.

(d) It prompts the operator to switch off the puller's power supply, and submits control to the analog controller when the operator indicates that this is possible by commanding "EXIT" again. The operation mode is set to Monitoring with a suitable command message.

(7) EXICZO disables the output of measured data, and stops the Measured Data Output Task MEASDO (compare chapter 4.4.5). Simultaneously, it resets the Timer Output Enable flags ENTIMO to prevent the display of new time strings.

(8) It closes all possibly still open output files,

(9) Clears the console screen and writes a sign-off message,

(10) Switches all output relays off, and

(11) Calls the routine FREXIT which will re-boot RXISIS-II.

#### <u>4.4.3.14 OVERLAY CZOV14 - MODULE CONDIT - COMMAND IF</u>

CONDIT receives the command input line buffer from COMINT; it tries to retrieve a Variable name from it by scanning to the first non-blank character after the first space. A Variable name is requested if none was found. This name is converted to uppercase (FRCVUC) and processed by FINDAD which returns the address and the type of the Variable specified. Next, one or two relational characters ("<", "=", or ">") are either read from the input buffer, or explicitly requested. A numeric value of 1 to 3 is assigned to each valid relational character; the two relational characters and the Variable type are packed into one byte of the command message in order to conserve space. After a comparison value which is simply stored in the command message, the name of the Macro command which is to be executed conditionally is retrieved in the standard way. A Macro file name is built from the command name (with MAKEFN), and CONDIT tests the requested Macro file in the same way which COMINT uses for the same purpose. The Macro name is stored in the command message which is dispatched if the file exists and the operator acknowledgement was obtained.

### 4.4.3.15 OVERLAY CZOV15 - MODULE DISPLY - COMMAND DISPLAY

DISPLY requires the name of the Variable whose value is to be displayed as its only input. The name is either read from the input line buffer, or explicitly requested. After a conversion to uppercase, it is submitted to FINDAD which returns the address and the type of the Variable. The Variable is read subsequently with the algorithms already discussed above (compare chapter 4.4.3.2), and displayed according to its type.

### 4.4.3.16 OVERLAY CZOV16 - MODULE DOCUMT - COMMANDS FILES AND DOCUMENTATION

The module DOCUMT is accessed during the initialization sequence, from FILES, and at a DOCUMENTATION command call. DOCUMT is very similar to REQCMF (compare chapter 4.4.3.10) and to DATAFI (compare chapter 4.4.3.12). The major differences between these routines are (aside from the different IOFLAG and FILLOC array elements which they use):

(1) DOCUMT explicitly permits to use the printer as an output device (which would not make sense with the other two files).

(2) It permits to specify an interval for the periodic output of measured data to the Documentation file, and

(3) It opens the Documentation file, enables printer output (in case it was disabled due to a printer timeout), and initializes the output routines with a call to the routine STARTP which is part of the DATOUT module (compare chapter 4.3.1). STARTP presets the line counter and generates a page header line in the Documentation file.

### 4.4.3.17 OVERLAY CZOV17 - MODULE DIRECT - COMMAND DIR

DIRECT displays the directory of the disk in the drive specified, together with some information about the disk itself. Having obtained a valid drive number (0 or 1), DIRECT first reads the disk label, i.e., the name of the disk, which is kept in the file ISIS.LAB. Next, the routine determines by checking the file location array FILLOC whether files are open for output on the specified dis[illegible] In this case, the information about the free and used disk space cannot be reliably obtained, and the corresponding values are preceded by "less than" and "greater than" signs, respectively. The number of occupied sectors on the disk is retrieved from the disk map file ISIS.MAP; each bit set in ISIS.MAP corresponds to a used sector. These bits are counted by the (assembly language) Function BITCNT, and written to the directory header line. Since each single-density, single-sided 8" disk holds 2002 sectors of 128 bytes each, the number of free sectors can be easily calculated. Finally, DIRECT reads the disk directory file ISIS.DIR, and displays all file names in a way similar to the one chosen for the Help menu output (compare chapter 4.4.3.4). DIRECT can only output 6 entries per screen line since full file names, including extensions, have to be displayed. The routine pauses after having written the header line and four lines of directory contents, and continues overwriting the directory display with new data after the operator pressed the "Enter" key. DIRECT returns to COMINT when the "Enter" key was pressed after the last valid directory entry was displayed.

## 4.4.3.18 OVERLAY CZOV18 - MODULE RESOVL - COMMAND RESET

The RESET command is indispensable for the initialization of the diameter evaluation routines. It prepares not only the buoyancy compensation routines in the module SHAPE (compare chapter 4.5.2) but initializes also the weight and length values displayed. RESOVL offers the standard option of resetting length and weight to zero; it permits to maintain the current value for each of these parameters or to enter new values if the zeroing option was rejected. The values input by the operator are scaled to obtain integer data in the formats used internally for weight and length representation; a value of -32768 (the most negative integer value) indicates that the corresponding parameter value should be preserved. RESOVL sends these values to the Command Executor which calls the actual reset routine.

## 4.4.3.19 OVERLAY CZOV19 - MODULE INIDAT - COMMAND INITIALIZE

INIDAT is called upon an INITIALIZE command and, in addition, during the system preparation sequence. It displays the current values of six system parameters (seed and crucible diameter, boric oxide weight, and the densities of the solid crystal, the semiconductor melt, and the boric oxide melt), and permits the operator to either accept them by pressing "Return" only, or to enter new data. Negative values, which are invalid in any case, are trapped, and the parameters are converted back to their internal storage format. In order to facilitate diameter evaluation, the system holds the squares of the diameter values, and densities in grams per cubic millimeter. Finally, INIDTA checks the minimum height of the boric oxide encapsulant melt (i.e., the height of a cylinder of molten boric oxide with the specified mass covering the entire cross section of the crucible), and sets the boric oxide weight to zero if it is too small to be handled properly by the Diameter Evaluation routine SHAPE (compare chapter 4.5.2.3).

## 4.4.3.20 OVERLAY CZOV20 - MODULE PLOTOV - COMMAND PLOT

The module PLOTOV permits to link Variables or memory locations specified by absolute addresses to one of the eight Plot Channels. An approach similar to the one used in DEBUGO is applied to separate Variable and address inputs; Variables must be in INTEGER*2 format in order to be displayed, whereas this format is intrinsically assumed for memory locations specified by their absolute addresses. PLOTOV scans the input string for name/address and channel information, and requests data if applicable. Further processing of the PLOT command is done by the Command Executor to which a pertinent message is dispatched.

## 4.4.3.21 OVERLAY CZOV21 - MODULE CLEARO - COMMAND CLEAR

Two versions of the CLEAR command are supported by CLEARO, namely, the Unconditional and the Selective Clear. CLEARO scans the input line for the name of a Variable, and assumes that an Unconditional Clear is issued if no valid data is found. An operator reconfirmation is requested, and a prompt for a Variable name is issued if the operator indicates he did not want an Unconditional Clear. The Variable name is processed as usual (with FINDAD), and a command message is sent to the Command Executor when the operator acknowledges his entries.

## 4.4.4 THE COMMAND EXECUTOR - TASK CMMDEX

The Command Executor receives command messages from two sources, namely, from the Command Interpreter, and from the Command File Input Task. The special Fortran-RMX-80 Interface Routines used (compare Appendix A) automatically advance these messages to the Command File Output Task which eventually records them in the Command Output file. The Command Executor's commission is to process each command message, and to perform several other procedures which have do be done in regular intervals.

CMMDEX runs once every second; its timing is indirectly derived from the Timer Task FXTIME. The first action of CMMDEX after it performed a few initialization subroutine calls (but not the first action after it starts running every second) is to receive a command message if there is one. In most cases, there will be none; the approach of using command messages has, however, the advantage that these messages will be queued by the operating system in the order in which they were issued if more of them are generated than can be processed. Therefore, it is possible to have CMMDEX process only one command message every second without losing commands; in the worst case, the command execution may be delayed by a few seconds.

#### 4.4.4.1 COMMAND MESSAGE PROCESSING

CMMDEX first decodes a command message if one was received. The first byte of each command message holds a command code which consists of a major Mode (corresponding to "SET" or "DEBUG") in the high four bits of the byte, and a Switch (e.g., for "SL" or "Continuously") in the low four bits. These values are separated, and control is vectored to the proper processing sequences.

**Mode = 1 and 2 - SET and CHANGE Internal parameter**

Mode values of 1 and 2 correspond to SET and CHANGE commands, respectively, which apply to Internal parameters. CMMDEX first determines the address where the specified setpoint is to be stored: There are two arrays for the setpoints of the Internal parameters, STPNT0 and STPNT1, which correspond to the left and right setpoint columns displayed on the console screen. STPNT0 always holds the setpoint values which are actually used by the various controller routines, and which are the ones which are normally ramped by CMMDEX. There is, still, an important exception to this rule if an Internal parameter is controlled, e.g., the heater temperatures in diameter controlled operation modes. In this case, CMMDEX stores the output of its ramping generator in STPNT1 rather then in STPNT0; the LOGICAL Function CNTRL (which is actually an assembly language subroutine) returns ".TRUE." in this case. CMMDEX sets a memory location according to the (setpoint) variable type (all Internal parameters are two-byte integers) and enters a sequence of code which is also used for SET and CHANGE commands applying to Variables, and by the Conditional Command Executor algorithms. A flag (L) is used to branch to the Conditional Command Executor code after the common sequence; although this approach is certainly not consistent with structural programming techniques, it is the most efficient way with regard to the large number of local memory locations whose contents are used inside and outside the common code.

Having issued a warning if a SET or CHANGE command was entered while the CGCS is in monitoring mode, i.e., not controlling the puller, CMMDEX determines the current contents of the target locations, and converts them to floating-point format if necessary. For CHANGE commands (Modes 2 or 10), a new final value is calculated by adding the message input value to the current target location contents; for SET commands, it is directly derived from the value passed with the command message. The magnitude of the resulting value is checked if it has to be stored to integer locations which have a limited numeric range; the result is set to the permitted maximum with the correct sign, and an error message is issued, if an overflow is detected. Similarly, diameter, temperature, and power limit setpoints are checked for negative values; the above result is set to zero and an error message ensues if any of these setpoint values is found to be negative.

The processing of all SET and CHANGE commands continues in a Command Executor code sequence called Ramping Preparation: The CGCS is able to ramp up to twenty independent parameters; all ramping control structures are therefore arrays with twenty elements each. Each channel holds an address, a variable type, and starting, final, increment, and breakpoint values; the latter four in floating-point notation to guarantee the necessary resolution and dynamic range. CMMDEX assigns a ramping channel according to availability to a parameter or Variable which is to be ramped; only the Variable address indicates which data is handled in which channel. In order to prevent confusions if a SET or CHANGE command was issued for data which is already being ramped, CMMDEX checks first whether the address passed is already used by one of the ramping channels, in which case this particular channel is updated. Otherwise, CMMDEX searches for an unused ramping channel (unless all channels are used or a transition time of zero was entered, in which cases ramping is bypassed, and the final value is immediately stored at the target address). The status of a ramping channel is determined by RMPFLG which is zero if a channel is not used. It is set to the number of an Internal parameter (1 to 9), or to -1 if a Variable is ramped. The Ramping Preparation code stores the current and final values of the data to be ramped; an increment value is calculated by dividing the difference between the initial and the final values by the ramping time in seconds, and a breakpoint, by multiplying the absolute value of the increment by 1.1 and adding a small number. Finally, CMMDEX stores the final values of Internal parameter setpoints in the corresponding array (unless the parameter is being controlled), and continues with the Ramping Executor sequence.

Mode = 3 - Macro command, Unconditional CLEAR

A Mode value of 3 indicates either a Macro command (Switch = 0), or an Unconditional CLEAR command (Switch = 1).

The Macro name passed with a Macro command message is expanded into a full Macro file name (i.e., ".CMD" is appended). A possibly active Macro command is preempted (with a QUITCM call), and a pertinent message is issued. CMMDEX tries to open the Macro file; the QUITCM call is repeated if the old Macro file was not yet closed by the Command File Input Task or if the file could not be opened due to a temporary shortage of pool memory (which may happen under adverse conditions), an explicit error message ("Macro ... doesn't exist") is generated if the new file was not found, and the internal disk error message is output in case of any other error. CMMDEX reads the first 16 bytes of the new Macro command file and checks whether the first two bytes hold zeros, and the next two, the version code of the currently used system. The Macro command is cancelled if the first condition is not met, and a message referring to a "restricted command set" is issued if the system versions do not match. The flag DEBUGE is set in this case; it indicates to the Command File Input Task that all commands which refer to absolute memory locations, i.e., all commands with a command Mode value greater than 7, must be discarded (compare chapter 4.4.6). In any case, IOFLAG(4) is set to indicate to the Command File Input task that there is an active Macro file, and the flag RUNTIM activates the alarm clock timer. An internal counter, MACPRO, is set to a starting value of 4. This value will be decremented by 1 during each of the subsequent passes of CMMDEX, once every second, until it finally reaches zero; the checking of Conditional commands is inhibited while MACPRO is not zero (compare chapter 4.4.4.5). Finally, CMMDEX writes the name of the Macro command into the top line of the output screen, and generates a message which indicates that the execution of this Macro was started.

The Unconditional CLEAR command is processed very simply: The eight Conditional Command flags and the Conditional Command counter are reset, and a pertinent message is issued.

<u>Mode = 4 - MODE</u>

CMMDEX outputs a "New mode:" message upon receipt of a MODE command, and triggers a Data Dump in the Documentation Output. Subsequently, it checks for the following mode changes which require special initializations:

(1) Change from "Monitoring" to any controlled mode: In order to avoid transients when the CGCS takes over from the analog system, all measured values of the Internal parameters have to be duplicated to the corresponding setpoint locations in STPNT0. Simultaneously, all ramping flags and the ramping counter are reset. It is therefore not possible to transfer a constant or ramped Internal parameter setpoint from uncontrolled to controlled mode.

(2) Change from not diameter controlled modes ("Monitoring", "Manual") to any diameter controlled mode: CMMDEX checks in this case whether the Diameter setpoint is currently being ramped, stops its ramping if it is, and copies the current actual Diameter value to both Diameter setpoint locations.

(3) Changes between modes in which certain Internal parameters are controlled: All twenty ramping channels are scanned to find a ramped Internal parameter which was not controlled in the previous mode but is controlled in the new mode, or vice versa. In the first case, the output of the Ramping Generator is directed to the second setpoint array STPNT1 rather than to the first one, STPNT0, and the current value of the affected element in STPNT0 is copied to STPNT1 in order to avoid transients; in the second case, the ramping is stopped.

CMMDEX finally sets the Mode flag proper and jumps to a sequence in the Ramping Executor which outputs the Mode information in the "fixed" part of the console screen.

<u>Mode = 7 - RESET</u>

A RESET command is processed by a simple call to the subroutine RESET which is part of the assembly language module SHAPE (compare chapter 4.5.2).

**Mode = 9 and 10 - SET and CHANGE Variable**

Essentially, the algorithms described for Mode = 1 and 2 are used in treating Variables.

**Mode = 11 - IF and Selective CLEAR**

A SWITCH value of 0 represents a Conditional (IF) command, a value of 1, a Selective CLEAR.

CMMDEX can handle up to eight Conditional Macro commands; Conditional commands which are issued while already eight commands are pending are ignored, and a pertinent error message is issued. After a free storage location was found for the new command, the two relation code values and the Variable type information are extracted from the command message byte in which they were stored, and the comparison value, the Variable address, and the name of the Macro command which is to be executed conditionally are written to the proper locations for use by the Conditional Command Executor.

In case of a Selective CLEAR command, CMMDEX compares the Variable addresses stored for all currently active Conditional Command Channels to the Variable address passed with the command message. A channel is deactivated (by resetting its status flag), and a "Conditional Macro cleared" message is issued if matching values are detected. (This may happen more than once if several Conditional commands referred to the same Variable.)

**Mode = 14 - PLOT**

CMMDEX stores the address passed with the command message in the element of an address array which is determined by the Plot Channel number specified with the command.

**Mode = 15 - DEBUG**

The DEBUG command uses a Switch value which can have the values 2 through 5. Upon a DEBUG Continuously command (Switch = 2), the variable address and type are stored in locations of the pertinent Debug arrays whose index is determined by the Debug Channel number (1 to 4). The DEBUG Modify command (Switch = 3) is processed by storing the correct number of bytes at the specified address. The DEBUG Resume and Suspend commands, finally, are executed by calls to the proper Interface routines.

### 4.4.4.2 THE RAMPING EXECUTOR

This part of the Command Executor is accessed after the treatment of any command, and if no command was received at all. The first commission of the Ramping Executor has nothing to do with ramping yet: CMMDEX tests the flag RESTDO(2), and writes the operation mode to the "fixed" part of the console screen whenever RESTDO(2) is found set, resetting the flag simultaneously. (This flag is set within the routine FRAME to enforce data output to the console screen after it was cleared; compare chapter 4.4.3.8.)

CMMDEX waits now for a so-called "Flag Interrupt" which is triggered once every second when a flag is set by the Analog Controller Task ANACNT which, in turn, is triggered directly by the Timer Task FXTIME. (The structure of the Interface software prohibits that several tasks be triggered in parallel by the Timer output.) The subsequent parameter ramping is therefore done in relatively regular intervals of one second, no matter how long the processing of input data took.

For each active ramping channel (ramping flag not equal to zero), CMMDEX tests whether the absolute value of the difference between the current setpoint and the final value is already less than the breakpoint value which was determined during the ramping preparation. The current setpoint is set to the final value in this case, and ramping of this channel is disabled. Otherwise, the increment is added to the current setpoint, and the current setpoint is stored in memory in the proper format in either case.

### 4.4.4.3 FLOATING-POINT CONVERSION OF MEASURED DATA

All measured analog data is primarily stored and processed as two-byte integers. Unfortunately, these values are hardly suitable for use in Conditional Macro commands because they have to be scaled to be meaningful. This is done by the Command Executor during each pass.

### 4.4.4.4 DEBUG DATA RETRIEVAL

During each pass, CMMDEX reads four bytes at the addresses specified with each active DEBUG Continuously command, and stores them in an array from which they will eventually be output by the Measured Data Output Task.

#### 4.4.4.5 CONDITIONAL COMMAND EXECUTOR

The Conditional Command Executor part of CMMDEX is only executed if the internal counter MACPRO holds zero. MACPRO is set whenever a new (Conditional or unconditional) Macro command has been started (compare chapter 4.4.4.1), and is reset to zero within several seconds. (The timing of CMMDEX is slightly corrupted when a Macro command is being activated, due to the relatively time-consuming disk accesses involved in this process. It is therefore not possible to specify the exact duration of the delay enforced with MACPRO.) Disabling Conditional command checking temporarily while a new Macro is being started guarantees that at least the first command of the new Macro can be executed (provided its relative time is zero or 1 seconds) without being preempted by a possibly concurrently activated Conditional command. (The first command in each file which must not be prematurely preempted should therefore be a CLEAR command at a relative time of 0 or 1.)

For each of the eight potentially pending Conditional Macro commands, the value held at the specified Variable address is read (using the algorithm of the SET/CHANGE commands, compare chapter 4.4.4.1) and compared to the constant passed with the command. The result of the comparison is compared to the specified relation(s), and the Macro command is invoked using the code sequence of the Macro command processing described in chapter 4.4.4.1. The message "Conditional Macro started" is output before control is transferred to the standard Macro command processing. This results in a possibly confusing sequence of commands if the Conditional Macro preempts an active Macro command:

"Conditional Macro started" - output by the Conditional Command Executor.

"Command Macro preempted" - refers to the old Macro.

"Executing Macro ... " - gives the name of the new Conditional Macro started.

#### 4.4.4.6 DATA DUMP TO THE DOCUMENTATION FILE

Data dumps are generated by the subroutine DUMPDT which is invoked by the Command Executor's main routine CMMDEX. DUMPDT enters its main body if either the Dump Flag DUMPFL is set, indicating a Data Dump request, or if the number of one-minute Flag Interrupts has been encountered which was specified as a Dump interval when the Documentation file was opened. DUMPDT generates three output lines which contain 21, essentially measured, data identified by two-character mnemonics (compare chapter 3.5). A pointer array is used to assign data from the (floating-point) array REALDT to the proper output locations, and output is written to a buffer which is pre-loaded with the identification text frame when the CGCS is loaded into memory. The three buffers each of which contains seven data values are written in turn to line 24 of the console screen (which is, in fact, not usable for permanent display since it is cleared during each console input request), using the standard Console/Documentation file output routines STROUT (compare chapter 4.3.1). In this case, it is not the output on the CRT screen we are interested in, but its duplicate, tagged with the time information, in the Documentation output. (Line 24 of the CRT screen is cleared anyhow by a concluding line of spaces to keep the console screen tidy.) Finally, DUMPFL is reset in any case in order to be ready to accept a new Dump request.

The Fortran module which holds DUMPDT contains, in addition, a small routine DUMP which can be called by any task which wants to trigger a Data Dump. In addition to setting DUMPFL to ".TRUE.", this routine sets the flag byte TIMINT which is normally set by the Timer Task FXTIME after the interval specified when the Data file was opened, triggering the output of a data record to the Data file. Therefore, an additional record is entered into the Data file when DUMP is called. (The status of TIMINT does not matter if no Data file is open and active.)

### 4.4.4.7 ANALOG OUTPUT TO A CHART RECORDER

The subroutine PLOTPR which is called as the next action of the Command Executor Task CMMDEX prepares data for analog output. PLOTPR does not output this data, though; the latter task is performed by the Analog Data Controller ANACNT.

First, PLOTPR calculates the "expanded" temperature, growth rate, and diameter and crucible position errors which were specially provided for chart recorder output. This procedure involves, in general, proper scaling, limiting to maximum and minimum values, and adding of an offset if required. Next, PLOTPR retrieves the contents of the eight locations pointed to by the Plot Channel address array elements. It calculates the absolute values of this data, and provides a message on the console (and in the Documentation output) if a Plot Channel changed its sign since the last pass of PLOTPR. The resulting eight INTEGER*2 values are stored with interrupts disabled in order to prevent problems caused by the real-time environment; the (assembly language) subroutines DISINT and ENINT disable and enable interrupts, respectively.

#### 4.4.4.8 PROGRAM CODE INTEGRITY CHECK

In order to improve the chances to detect inadvertent modifications of the CGCS program code - due to hardware failures or to software problems -, the routine MEMCHK is called at the end of the CMMDEX code, before the task resumes its infinite loop. MEMCHK calculates a signature byte for each 256 byte page within the main CGCS program code, and compares it to the signature obtained during the previous pass. An error message is issued if the two signatures are found to be different, i.e., if one of the 256 bytes checked changed its contents, and the signature byte kept in memory is set to the new value. The output of error messages is suppressed during the first pass of MEMCHK, immediately after the CGCS was loaded, to permit the array of signature bytes to be initialized properly. With each call of MEMCHK, five (5) 256-byte pages are processed; MEMCHK loops to the beginning of its surveillance area after it arrived at its end. The about 150 memory pages which are monitored by MEMCHK are therefore tested once every 30 seconds. The memory check comprises the entire CGCS resident code area; for obvious reasons, it could neither include the data locations nor the overlay area. Still, the memory check encompasses about 70 percent of the entire memory area, and it is easily possible that MEMCHK might detect a damaged program code byte before it has been executed (with conceivably disastrous results).

### 4.4.5 THE MEASURED DATA OUTPUT TASK - TASK MEASDO

The task MEASDO provides all periodically updated output to the console. It is not synchronized to any other system task but loops continuously through its code. In order to prevent MEASDO from monopolizing the system (which would have the effect that tasks with a lower priority could never be executed), there are deliberate waits built into the task: At eight roughly equidistant locations within MEASDO, the task calls the subroutine WTOUTP which, in turn, executes a wait operation for a specifiable number of iRMX-80 time units. (One iRMX-80 time unit is 50 ms.) The number of time units for which WTOUTP waits is kept in the system Variable INTRVL which can be modified with SET, CHANGE, or DEBUG Modify commands. The smaller this number is, the faster runs MEASDO obviously;

the minimum value of INTRVL is 1. (An INTRVL value of 0 halts MEASDO indefinitely and irreversibly.) The current default INTRVL value is 10; this value gives satisfactory response during normal system operations. It is, however, recommendable to reduce the INTRVL value during adjustments of the analog data acquisition hardware. The INTRVL value should be restored to its standard value when full growth control is required, in order to protect the system from being overloaded.

The infinite task loop of MEASDO is entered after two initialization calls to Interface routines; it starts with a check of the flag RESTDO(3) which "short-circuits" the task if it is set, thus inhibiting data output. Next, MEASDO copies all data which is to be output into internal memory locations. This is necessary because most of the data locations are protected by software interlocks to prevent them from simultaneous accesses by several tasks. MEASDO would unduly block the data locations and, in consequence, all tasks which also attempt to access them if it monopolized them during the lengthy output operations; on the other hand, repeated access and release operations would impose an unacceptable overhead. The approach chosen for copying the data locations one by one, rather than using a program loop, may bewilder experienced Fortran programmers. For the given number of items, however, the technique chosen is faster and more code efficient than a loop.

The actual output operations follow one standard approach: The current (integer) data value to be output is compared to the value of the same item during the previous pass of MEASDO. The integer value is multiplied by its scaling factor and written to the console only if the two values are different, thus preventing the repeated output of constant data. In order to enforce the data output regardless of whether an item was updated or not, a loop is provided which sets all items of the "old" array to the contents of the corresponding items in the "current" array, plus 1, if the flag RESTDO(1) is set. Evidently, the two arrays will hold different values for each output item after this procedure.

After having output all items, MEASDO sets the "old" output data array to the contents of the "current" data, and tests whether output is required for DEBUG Continuously commands. The scrolled output area is limited to four lines (18 through 21), and line 17 is cleared if Debug output was activated since the last pass; the scrolled area is set to five lines (17 through 21) if Debug output was deactivated. MEASDO writes the address and the memory contents for each active Debug channel into line 17; prior to the actual data output, the screen area corresponding to the respective channel is cleared (overwritten with spaces) if the Debug output mode was changed. This procedure clears channels which are deactivated, and it removes the previous output completely if the interpretation of Debug data and therefore the number of displayed digits was changed.

## 4.4.6 THE COMMAND FILE INPUT TASK - TASK CMFINP

The Command File Input Task CMFINP reads commands from a Macro file and sends them with the proper timing to the Command Executor.

Each command in a Macro file is tagged with the time relative to the call of the Macro command at which the particular command is to be executed. This is accomplished by a combined action of the Command File Input and the Timer Tasks:

The Command Executor sets the flag RUNTIM after each call to a valid Macro command file, which indicates to the Timer Task that a second seconds counter is to be started. The contents of this counter are compared by the Timer Task once every second to an "alarm clock" value, and a flag is set if the counter value is equal to or greater than the "alarm clock" setpoint. The "alarm clock" value is set by the Command File Input task according to the relative time of the next command which is to be processed; CMFINP is, in turn, triggered by the "alarm clock" flag.

The task loop of CMFINP starts with a wait for an "alarm clock" flag interrupt. Since CMFINP set the "alarm" value to zero, such an interrupt will immediately happen when the timer is enabled by CMMDEX upon a Macro command. CMFINP reads one record from the Macro command file. The task branches if a disk error occurred, if the end of the file was encountered, or if a command which refers to absolute memory locations was read from a Macro file generated under a different system version. In the first two cases, an "End of Macro command file" message is output, the input from the file is disabled (IOFLAG(4) is reset), the Macro file is closed, the Macro name in the first line of the console screen deleted, and the second timer disabled by resetting RUNTIM (which is called INPACT in CMFINP). CMFINP reports "Macro command not executable" and reads the next command if the third exception condition was detected. For all valid commands, CMFINP sets the "alarm clock" to the execution time of the next command, and loops back to the initial wait. The command message is dispatched to the Command Executor immediately after the "alarm clock" interrupt.

This handling of Macro command files accounts for possibly confusing sequences of system messages: CMFINP reads the next command immediately after having sent the preceding one to the Command Executor. CMFINP messages may therefore appear on the screen even before the last command was processed by the Command Executor.

### 4.4.7 THE COMMAND FILE OUTPUT TASK - TASK CMFOUT

The Command File Output Task CMFOUT receives command messages from the Command Executor. It appends the relative time of the command, i.e., the difference between the current value of the seconds counter in the Timer Task and the value which this counter had when the START command was issued. This time information is stored in the first two bytes of a 16 bytes record, followed by the command message proper. CMFOUT traps Macro commands which are not recorded on purpose (compare chapter 4.4.3), writes the record to the Command Output file, and disables itself if the command was an END command.

### 4.4.8 THE DISK OUTPUT TASK - TASK DSKOUT

The task DSKOUT which resides in the Fortran module DSKDAT is in charge of output to the Data file. DSKOUT collects all data which is to be recorded in a buffer which corresponds already to the future contents of the disk data record, and writes this buffer to the disk.

## 4.5 THE PROCESS CONTROLLER

### 4.5.1 THE PID CONTROLLER ROUTINE FRPIDC

The actual process control approach used in the digital CGCS is, to a large degree, based on conventional analog techniques. In particular, the system uses PID (Proportional-Integral-Derivative) controllers for the closed-loop control of various parameters. In contrast to an analog system, however, where separate controller hardware is required for every control loop, the CGCS contains only one generic PID controller routine which performs (with different parameters) the following functions:

(1) Motor Speed Control: The outputs which determine the speeds of the four puller motors (seed and crucible lift and rotation) are controlled according to the differences between the corresponding setpoints and the actual speed values. This approach permits to compensate for motor controller imperfections such as nonlinearities or offsets.

(2) Temperature Control: The power output by the heater(s) is controlled to maintain the heater temperature setpoint(s).

(3) Diameter Control: The heater temperature setpoints are adjusted to provide a minimum diameter error.

(4) Crucible Position Control: The lift speed of the crucible is controlled to reduce to zero the difference between a calculated crucible position setpoint and the pertinent actual value.

The PID Controller routine is based upon high-speed integer algorithms. Its output is calculated in several steps: Within the first step, the error E is derived from the setpoint S and the actual value A by:

$$E = S - A \qquad (1)$$

Subsequently, an intermediate result X is calculated according to the following algorithm:

$$X = E*P/256 + (IE*I)/IS + DE*D/256 \qquad (2),$$

with P, I, and D, the proportional, integral, and derivative multipliers, respectively. IE represents the error integral, i.e., the sum of all error values encountered since the PID controller went into operation; IS is a scaling factor which can be either equal to 256 or to 65536 ($2^8$ and $2^{16}$, respectively), depending on the value of a control flag. DE, finally, is the difference to the preceding error:

$$IE*I = E(0)*I + E(1)*I + \ldots + E(n)*I \qquad (3)$$

$$DE = E(n) - E(n-1) \qquad (4)$$

The three terms in (2) are scaled by 256 (or by 65536) in order to permit effective proportional, integral, and derivative multipliers with an absolute magnitude of less than one. (The above values of the scaling factors were chosen because a multiplication or division by a power of two imposes the least time and code overhead.)

The integral component is calculated by accumulating the sum of the error values, each multiplied by the integral multiplier, in a four byte (32 bit) memory location. (This approach is less sensitive to abrupt changes of the integral multiplier I which may happen during the tuning of the system, compared to accumulating the error sum and multiplying it by the integral multiplier.) Depending on the magnitude of the integral scaling divisor IS, either the least significant byte, byte 0 (for IS = 256), or the two least significant bytes, bytes 0 and 1 (for IS = 65536), of this internal sum are discarded when the integral component of X is determined. The integral component of X is the value of the next two bytes, 2 and 1, and 3 and 2, respectively, which is rounded according to the most significant bit of the discarded byte(s), and set to the maximum positive or negative value if IS = 256 was chosen and the accumulation of the integral extended into the fourth byte (byte 3). Optionally, the resulting two byte integral component may be compared to a limit value; the entire four byte integral is modified to return an integral component which is exactly equal to the limit value (with the sign of the four byte error integral) if the integral component would otherwise exceed the limit.

The intermediate result X is calculated by first adding the proportional and the derivative components, and finally, the integral component. X may be limited to any arbitrary range if the user chooses so; for a given limit L, X results in

$$\begin{aligned} X &= -(L+1) && \text{if } X < -(L+1) \\ X &= L && \text{if } X > L \\ X &= X && \text{otherwise} \end{aligned} \qquad (5)$$

This limiting operation can be selected independent from the limiting of the integral component although the same limit parameter is used. A default value for L is assumed in either case (with L = 32767) if either no limit was specified, or if a negative limit value was given.

In order to improve the dynamic response of the PID routine, a "wind-up protection" feature was included. This feature prevents the error integral IE from overflowing when a limit condition is incurred. Without "wind-up protection", the error integral would continue accumulating in this case, which might become particularly disturbing if a scaling factor IS of 256 is used and the integral extends into the highest byte. The error integral would subsequently require a very long time to recover from the previous condition even if the error already changed its sign, which might obviously lead to control instabilities. The "wind-up protection" can be explicitly activated by the user; it becomes only effective when a limit condition is incurred. In this case, the internally stored four byte error integral can be adjusted in either one of two ways: it can either be set to a value resulting in a (two byte) integral component equal to the difference between the limit value and the sum of the proportional and the derivative components (Mode A), or it may be adjusted to return an integral component equal to the positive or negative limit value, as demanded by the error integral's sign (Mode B). Thus, the PID controller is forced to remain in its active operation area; the intermediate result X reacts immediately or almost immediately to a decrease of the error rather than after the delay otherwise inherent with the reduction of the error integral. Activating the "wind-up protection" overrides the integral component limiting function.

Finally, the intermediate result is submitted to two additional adjustments: First, it is multiplied by a factor which is a (positive or negative) power of two, and second, a bias value B is added:

$$M = B + 2^{**}G * X \qquad (6)$$

The result, M, is output at last. The scaling factor 2**G was provided to permit an adjustment of the controller's output to miscellaneous devices. Some devices require, e.g., less than 16 bit signed data, in which case a negative G value can be used to dispose of the least significant G bits. It could also be used for restricting the output signal to a certain range. A positive G could increase the overall gain of the controller; with regard to accuracy, this is, however, not recommended. The bias value B, finally, centers the output of the PID controller around the bias value.

This handling of the bias value permits to introduce a nonlinear PID controller response by means of two stacked PID controllers. The setpoint and actual data inputs of both controllers are to be connected in parallel; the output of the first is used as a bias input for the second. Single controller operation can be achieved by setting all multipliers of one of the two stacked controllers to zero. (Which one does not matter since they are exchangeable.) Nonlinear control is possible if different parameters are attributed to both controllers and the output limit L is set to a value considerably less than 32767 for one of them. For an error resulting in an intermediate signal X of the output limited controller less than ±L, the output of the two controllers is the sum of the outputs of either controller, and the resulting P, I, and D values are the arithmetic sums of the corresponding parameters of both controllers. In the output limited mode, the limited controller contributes only its limit value whereas the other controller continues operating in its linear range; the P, I, and D parameters of the two-controller system are thus equal to the parameters of the controller remaining active. It seems, however, advisable to introduce some kind of wind-up protection at least for the output-limited controller in order to permit a fast response of the system to sudden error changes. The advantages and disadvantages of the approaches feasible with FRPIDC are discussed in Appendix J.

Essentially, the operation of the digital PID controller routine is akin to any analog PID controller. The time constants in the integral and derivative parts of the controller function are determined by the frequency 1/T with which the controller runs. In the linear region of the controller (no limit incurred), eqs. (2) through (6) may be re-written as:

$$M = B + 2{**}G * P'[E + I'/(P'{*}T) \int_0^T Edt + (D'{*}T)/P'{*}dE/dt\ ] \quad (7)$$

with P', I', and D', the proportional, integral, and derivative multipliers of eq. (2) times their appropriate scaling factors.

The parameters for the PID controller are kept in a 12 byte array. The first two bytes of this array must be accessed from Fortran as INTEGER*1 locations (one byte integers), the remainder, as INTEGER*2. The following data is kept in this array:

```
└─────────┘               Byte 0: Gain Multiplier Exponent G

└─────────┘               Byte 1: Control Byte, containing switches
    ││││                          for:
    │││└Integral Component Scaling: 0 ... IS = 256
    │││                              1 ... IS = 65536
    ││└Output Limit: 0 ... No Explicit Output Limit
    ││                1 ... Output Limit = +/- L
    │└Wind-Up Protection: 0 ... Off
    │                     1 ... On
    └Wind-Up Protection Mode: 0 ... Integral set to L-(P+D)
                              1 ... Integral set to ±L
    Integral Component Limiting: 0 ... Off
                                 1 ... On

└─────────┴─────────┘     Byte 2+3: Bias Value B

└─────────┴─────────┘     Byte 4+5: Proportional Multiplier P
```

Byte 6+7: Integral Multiplier I

Byte 8+9: Derivative Multiplier D

Byte 10+11: Limit Value L

The control byte permits to set the operation mode of the PID routine, namely, the scaling of the integral component, the output limiting operations, and the wind-up protection. The decimal values of the control byte listed below correspond therefore to the following operations:

| CNTL | IS | Limit | Wind-Up Prot. | | Integr.Lim. |
|---|---|---|---|---|---|
| 0 | 256 | ±32767 | OFF | | ±32767 |
| 1 | 65536 | ±32767 | OFF | | ±32767 |
| 2 | 256 | ±L | OFF | | ±32767 |
| 3 | 65536 | ±L | OFF | | ±32767 |
| 4 | 256 | ±32767 | ON | Mode A | ±32767 |
| 5 | 65536 | ±32767 | ON | Mode A | ±32767 |
| 6 | 256 | ±L | ON | Mode A | ±32767 |
| 7 | 65536 | ±L | ON | Mode A | ±32767 |
| 8 | 256 | ±32767 | OFF | | ±32767 |
| 9 | 65536 | ±32767 | OFF | | ±32767 |
| 10 | 256 | ±L | OFF | | ±32767 |
| 11 | 65536 | ±L | OFF | | ±32767 |
| 12 | 256 | ±32767 | ON | Mode B | ±32767 |
| 13 | 65536 | ±32767 | ON | Mode B | ±32767 |
| 14 | 256 | ±L | ON | Mode B | ±32767 |
| 15 | 65536 | ±L | ON | Mode B | ±32767 |
| 16 | 256 | ±32767 | OFF | | ±L |
| 17 | 65536 | ±32767 | OFF | | ±L |
| 18 | 256 | ±L | OFF | | ±L |
| 19 | 65536 | ±L | OFF | | ±L |
| 20 | 256 | ±32767 | ON | Mode A | * |
| 21 | 65536 | ±32767 | ON | Mode A | * |
| 22 | 256 | ±L | ON | Mode A | * |
| 23 | 65536 | ±L | ON | Mode A | * |
| 24 | 256 | ±32767 | OFF | | ±L |
| 25 | 65536 | ±32767 | OFF | | ±L |
| 26 | 256 | ±L | OFF | | ±L |
| 27 | 65536 | ±L | OFF | | ±L |

| CNTL | IS | Limit | Wind-Up Prot. | | Integr.Lim. |
|---|---|---|---|---|---|
| 28 | 256 | ±32767 | ON | Mode B | * |
| 29 | 65536 | ±32767 | ON | Mode B | * |
| 30 | 256 | ±L | ON | Mode B | * |
| 31 | 65536 | ±L | ON | Mode B | * |

Wind-up Protection Mode A entails that the integral component is set to the limit value minus the sum of the proportional and the derivative components if the output exceeds the limit; in Mode B, the integral component is set to the positive or negative limit value, as appropriate.

* Wind-up protection overrides integral limiting.

An analysis of the various limiting and anti-windup approaches available with FRPIDC is given in Appendix J.

### 4.5.2 THE DIAMETER CONTROLLER - TASK DIACNT

#### 4.5.2.1 THE DIAMETER CONTROLLER ROUTINE PROPER - MODULE DIACNT

The Fortran module DIACNT constitutes the main routine of the Diameter Controller Task. This task is triggered every ten seconds by a "flag interrupt" generated by the Timer Task.

DIACNT uses a command message of its own in order to perform automatic RESET and MODE commands. The commands issued by DIACNT are <u>not</u> recorded in the Control Output file, and the command message issued by DIACNT is returned to this task after having been processed by the Command Executor. This implies that DIACNT has to retrieve this message from its response exchange before it can be permitted to use it again. The FXACPT call at the beginning of the infinite loop in DIACNT performs exactly this task.

A sequence immediately following this subroutine call checks for mode changes into Diameter Controlled while the Diameter Evaluation routines have not been reset yet. The following steps ensue if such a condition is detected:

(1) DIACNT issues a RESET command which sets the weight and crystal length grown locations to zero, and generates a pertinent message.

(2) It stores the current MODE value (which can be 2, 3, or 4) in an auxiliary location INTMOD, sets the operation mode

to Manual, and marks this condition with a SHSTOL value of -3. The remainder of the task loop is skipped.

This procedure triggers a Reset operation when the Command Executor runs the next time; the actual diameter value is still meaningless because the Diameter Evaluation routine runs only <u>after</u> a Reset, but it will be ignored because the operation mode is still set to Manual.

(3) During its next pass, ten seconds later, DIACNT will set the MODE value back to the value saved in INTMOD; it will duly execute the Diameter Evaluation routines which will return a meaningful diameter value now, but it will skip the Diameter Control sequence.

The Command Executor runs only after DIACNT has finished (because its priority is much lower), and it will find a meaningful Actual Diameter value which it can copy to the Diameter Setpoint locations (because of the mode change to Diameter Controlled).

(4) Only at the third pass, twenty seconds after a mode change to Diameter Controlled without preceding Reset was detected, DIACNT will resume its standard operation.

Under regular operating conditions, DIACNT copies the operation mode value into a local location in order to avoid confusions if the mode is changed while this task is running.

DIACNT retrieves now from the array of analog input data the Crucible Position and the Differential Weight values, and converts the latter into floating-point notation, scaling it with the proper scaling factor. This data is first submitted to the subroutine ANOMAL (compare chapter 4.5.2.2) which performs an anomaly compensation if required, and subsequently, to the function SHAPE (compare chapter 4.5.2.3). SHAPE calculates a Diameter value (returned in DIAMET), and, in addition, the Length grown (scaled with the same factor as the Seed Position input data), and a Crucible Position setpoint (in SCRUCP) which is in the same format as the Crucible Position input data. (Several other auxiliary values are returned by SHAPE which are primarily intended for testing and debugging purposes.) SHAPE provides a status value in SHSTAT which is evaluated subsequently; corresponding (error) messages are issued the first time a certain SHSTAT value is returned, and the operation mode is set to Manual if either a Zero Seed Lift Speed or a Speed Overflow error was detected. With the exception of changes to or from a Zero Seed Lift Speed condition and of an Oxide Height Overflow, a Data Dump to the Documentation output and an additional record in the Data file are triggered. DIACNT tries to re-activate SHAPE after a Speed Overflow error with a call to the subroutine REACTV which is part of the assembly language module holding SHAPE; after six succeeding unsuccessful attempts, DIACNT decides that the problem is too serious to deal with it on its own, and disables SHAPE permanently (until a RESET command is issued again).

Subsequently, DIACNT enters the actual diameter controller sequence: While in Monitoring or in Manual mode, DIACNT has nothing to control. The routine resets, however, the Error Integral locations of the PID Data arrays and the Previous Error values (compare chapter 4.5.1) to zero. This is done to provide a defined environment when diameter control is activated.

In any one of the Diameter Controlled modes (Diameter, Diameter/ASC, and Automatic), DIACNT has to generate three Heater Temperature setpoints. Each of these setpoints is obtained from two stacked PID controllers which permit to obtain a nonlinear control response (compare chapter 4.5.1). The first PID controller receives the proper current Heater Temperature setpoint (i.e., the value obtained from operator or Macro commands) from the STPNT1 array as a Bias value; its output is used as a Bias for the second PID controller. All six Diameter controllers use the actual Diameter and the Diameter setpoint as inputs. The output of the second PID controller of each Heater channel is stored in the setpoint array STPNT0.

An additional control loop is executed in Automatic mode: The crucible lift speed is controlled according to the difference between the actual Crucible Position and the pertinent setpoint calculated by SHAPE. Two stacked controllers are used for this commission, too; similar to the Diameter controllers, the Crucible Lift Speed setpoint input by the operator (or from a Macro file) is used as a Bias value for the first controller, whose output is fed to the Bias input of the second controller. The second controller's output is stored as an actual Crucible Lift setpoint.

The approach chosen for the PID controllers in DIACNT, namely, passing the "manual" setpoint through the controller routines via the Bias inputs, has various advantages: Since the "manual" setpoint can be chosen to lie close to the actually required controller output, the PID controllers need only make small modifications to the "manual" setpoint, which improves the accuracy and the dynamic response of these routines. Furthermore, it is possible to limit the output of the controllers to lie within a relatively small range around the Bias value. This prevents, for example, the Diameter controller from totally turning off the heater if the actual crystal diameter seems to be much smaller than the pertinent setpoint, which may easily happen particularly during cone growth. In fact, a smooth transition from manual to diameter controlled growth may be obtained if the controllers' PID parameters are ramped from zero to their final values, or if the Limit values are initially set to zero and slowly ramped to their intended final values. There is, indeed, hardly any limit set to the control schemes which may be obtained from dynamically modifying the parameters of the controllers provided.

### 4.5.2.2 ANOMALY COMPENSATION - ROUTINE ANOMLY

Prior to the evaluation of the diameter, the Differential Weight value derived from the A/D converter can be submitted to a compensation for anomaly effects. According to the conventional anomaly compensation approach, a corrected Differential Weight X can be calculated from the "raw" weight Y by solving the differential equation

$$X = (Y - b \cdot X')' \qquad (1)$$

where X' is the first derivative of X with regard to time. Equation (1) can be re-written as

$$b \cdot X'' + X = Y' \qquad (2).$$

In the CGCS, we expanded the above approach to:

$$b \cdot X'' + a \cdot X' + X = Y' \qquad (3)$$

Numerically, the above differentiations have to be replaced by differences. With $X_0$, the current Differential Weight, $X_1$, the previous value, and $X_2$, the previous but one, we can write:

$$X_0' = X_0 - X_1$$

$$X_1' = X_1 - X_2 \qquad (4)$$

$$X_0'' = X_0' - X_1' = X_0 - 2 \cdot X_1 + X_2$$

Substituting eqs. (4) into eq. (3) results in a linear equation for $X_0$ which can be solved as:

$$X_0 = \frac{Y' + (a + 2 \cdot b) \cdot X_1 - b \cdot X_2}{1 + a + b} \qquad (5)$$

The "raw" Differential Weight Y' is input directly from the analog differentiator circuitry. A corrected Differential Weight $X_0$ can be calculated from the "raw" value Y and the previous results $X_1$ and $X_2$ according to (5). The Fortran subroutine ANOMAL evaluates $X_0$ from eq. (5) if the Mode value is greater than 2 (i.e., in Diameter/ASC and Automatic modes); otherwise, it sets $X_0$ equal to Y'. In addition, ANOMAL stores its $X_1$ value in $X_2$, and the $X_0$ value thus determined, in $X_1$, in order to have proper previous results for the next pass.

## 4.5.2.3 DIAMETER EVALUATION ALGORITHMS - ROUTINE SHAPE

The assembly language routine SHAPE constitutes the heart of the diameter evaluation algorithms. SHAPE calculates the following data:

(1) A crystal Diameter value which is derived from the Differential Weight which previously may have been submitted to anomaly compensation. SHAPE takes into account the buoyancy in the boric oxide melt.

(2) The current height of the boric oxide melt in the crucible.

(3) The Crystal Length grown.

(4) A Crucible Position setpoint which is used for determining the crucible lift speed in Automatic mode.

When the length of a crystal grown increases by $\delta x$, a portion of the crystal whose length is $\delta y$ emerges from the boric oxide melt. The two differential lengths are not necessarily equal since the height h of the boric oxide melt may have been changed by $\delta h$ due to a change of the crystal volume immersed (compare Fig. 10). We can write:

$$\delta y = \delta x - \delta h \tag{6}$$

The height h of the oxide melt can be determined from the oxide melt volume $V_m$ and the volume $V_i$ of the immersed part of the crystal, with R, the radius of the crucible:

$$V_m + V_i = R^2 \cdot \pi \cdot h \tag{7}$$

During the major part of a crystal growth run, $V_m$ is constant. Towards the end of the run, however, the semiconductor melt starts retracting from the crucible wall, resulting in a disk of molten gallium arsenide in the center of the crucible whose height remains roughly constant but whose diameter decreases. The gap between this disk and the crucible is filled by boric oxide, causing the effective oxide volume (i.e., the volume measured from the extension of the top surface of the semiconductor melt disk upwards) to decrease. Differentiation of eq. (7) results in:

$$\delta V_m + \delta V_i = R^2 \cdot \pi \cdot \delta h \tag{8}$$,

with

$$\delta V_m = - \epsilon \cdot \delta V_2 \cdot (d_x / d_{xm}) \tag{9}$$

and

$$\delta V_i = \delta V_2 - \delta V_1 \tag{10}$$,

where

$$\delta V_2 = r_2{}^2 \cdot \pi \cdot \delta x \tag{11}$$

and

$$\delta V_1 = r_1{}^2 \cdot \pi \cdot \delta y \tag{12}$$.

The parameter $\epsilon$ in eq. (9) is equal to zero during regular growth, and equal to 1 when the melt contraction has started. The effective boric oxide volume is reduced in this case by the volume of semiconductor melt required to grow the differential cylinder $\delta V_2$; $d_{xm}$ and $d_x$ stand for the densities of molten and solid semiconductor material, and $r_1$ and $r_2$ are the radii of the crystal at the oxide surface and the melt-crystal interface, respectively. With $d_o$, the density of the boric oxide melt, the change of the crystal's weight $\delta W$ can be written as:

$$\begin{aligned}\delta W &= \delta V_2 \cdot (d_x - d_o) - \delta V_1 \cdot (d_x - d_o) + \delta V_1 \cdot d_x = \\ &= \delta V_2 \cdot (d_x - d_o) + \delta V_1 \cdot d_o \end{aligned} \tag{13}$$,

The differential cylinder close to the semiconductor melt contributes to the weight only with the difference of the crystal and oxide densities, due to buoyancy; the differential cylinder which emerged from the oxide melt had previously a weight proportional to $(d_x - d_o)$ which is now proportional to $d_x$ only.

With eqs. (6) and (8) to (12), we can express $\delta y$ as a function of $\delta x$:

$$\delta y = \frac{R^2 - \beta \cdot r_2^2}{R^2 - r_1^2} \cdot \delta x \qquad (14),$$

with

$$\beta = 1 - \epsilon \cdot (d_x/d_{xm}) \qquad (15).$$

Note that $\beta$ is equal to 1 during regular growth, and it approaches 0 when the melt recession starts (because the ratio of densities is close to 1). With eq. (14), we can re-write eq. (13):

$$\frac{\delta W}{\pi \cdot \delta x} = r_2^2 \cdot (d_x - d_o - \beta \cdot d_a) + R^2 \cdot d_a \qquad (16),$$

with an "adjusted oxide density" $d_a$

$$d_a = d_o \cdot \frac{r_1^2}{R^2 - r_1^2} = d_o \cdot \frac{1}{(R^2/r_1^2) - 1} \qquad (17).$$

Eq. (16) permits to calculate the square of $r_2$:

$$r_2^2 = \frac{\frac{\delta W}{\pi \cdot \delta x} - R^2 \cdot d_a}{d_x - d_o - \beta \cdot d_a} \qquad (18)$$

With the Differential Weight ($\delta W/\delta t$) and the Growth Rate

$$v = \delta x/\delta t \qquad (19),$$

we finally can write for eq. (18):

$$r_2^2 = \frac{\frac{(\delta W/\delta t)}{\pi \cdot v} - R^2 \cdot d_a}{d_x - d_o - \beta \cdot d_a} \qquad (20)$$

The Growth Rate v is determined by the combined effects of the Seed Lift Speed $v_s$ and the speed $v_d$ with which the gallium arsenide melt drops:

$$v = v_s + v_d \qquad (21).$$

For a length $\delta x$ of crystal grown, the semiconductor melt in the crucible will drop by $\delta z$ in order to provide the crystal mass solidified while the crystal is within the regular growth regime. The melt level will hardly drop any more when the melt contraction towards the end of the growth run started. Since the total mass must be constant, we can write:

$$R^2 \cdot \pi \cdot d_{xm} \cdot \delta z = r_2{}^2 \cdot \pi \cdot d_x \cdot (\delta x + \delta z - v_c \cdot \delta t) \qquad (22),$$

with $d_{xm}$ and $d_x$, the densities of the semiconductor melt and the solid crystal, respectively, and $v_c$, the actual Crucible Lift Speed. We can solve eq. (22) for $\delta z$ and can, finally, obtain:

$$v = \frac{v_s - v_c}{1 - \alpha \cdot \dfrac{r_2{}^2 \cdot d_x}{R^2 \cdot d_{xm}}} \qquad (23).$$

The constant $\alpha$ is equal to 1 if (23) is obtained as an exact solution of eq. (22). In a heuristic way, however, assigning values different from 1 to $\alpha$ can help to compensate for non-ideal effects caused by the crucible shape and/or surface tension. Assigning values greater than 1 to $\alpha$ could compensate for a decrease of the crucible diameter close to its bottom; in contrast, $\alpha$ could be set to values less than 1 to take into account the receding of the gallium arsenide melt during the final stages of the growth process, an effect which obviously more than compensates for the beveling of the crucible. The surface of the melt does, in effect, not drop any more when the semiconductor melt starts receding from the crucible walls because the material used up by the growing crystal is supplied by reducing the melt's diameter rather than height. This corresponds to a crucible with infinite diameter, or to an $\alpha$ value of 0. At the end of the body growth, $\alpha$ may simply be ramped down to 0, starting at the point when melt recession usually begins. (A Variable named ALPHA is provided for this purpose. It is initialized with the value 1 but may be modified with the standard SET or CHANGE commands.)

The constant $\epsilon$ defined in eq. (9) follows exactly the opposite behavior, compared to $\alpha$: it is equal to zero during the regular growth, and assumes a value of 1 at the end of the process. It appeared therefore to be a reasonable approach to set

$$\epsilon = 1 - \alpha \qquad (24)$$

within the SHAPE software.

Obviously, the currently grown crystal diameter can be determined as twice the square root of the left side of eq. (20). SHAPE returns an INTEGER*2 value in the Variable IDIAMT; this value is converted to floating-point notation, scaled properly, and stored in the Variable DIAMET by the Command Executor.

The evaluation of eq. (20) implies a division by the Growth Rate v. Obviously, this value must not be equal to zero to permit the calculation of the diameter. SHAPE checks therefore the difference of the seed and the crucible lift speeds, $v_s$ and $v_c$, respectively, right at the beginning of its operations; SHAPE skips the remainder of its code and provides an error flag if it detects a zero value. The error flag is monitored by DIACNT; suitable actions are taken (compare chapter 4.5.2.1), and an appropriate message is output in the case of an error.

In order to solve eq. (20), the square of the radius of the crystal at the surface of the oxide melt, $r_1{}^2$, has to be known. This entails that the actual height of the boric oxide melt, h, and the total volume of the crystal immersed in the boric oxide, $V_i$, are also known; the latter parameters are required for calculating the optimum crucible position. SHAPE determines this data by keeping a table of crystal diameter squares in an array DIATAB. (In fact, SHAPE operates with diameter rather than radius squares; with the exception of a factor of 4 in the denominator of the first term in the numerator of eq. (20), the algorithms within SHAPE are identical to those above. We used radii rather than diameters in the above derivation in order to avoid naming confusions with the densities.)

Since SHAPE can only store the shape (i.e., the diameter) of the crystal at discrete length positions, an interpolation approach had to be developed which permits an approximate evaluation of the crystal's diameter at any arbitrary position. An obvious method would have been a linear interpolation between the stored diameter values. For the application in mind (where the squares of diameter values are more often required than the plain diameters), a linear interpolation of the <u>squares</u> of the diameter data proved to be considerably more efficient. Assuming that the <u>square</u> of the diameter or the radius is a linear function of the position x within the crystal, we can write

$$r^2 = k \cdot (x + x_o) \qquad (25),$$

with k and $x_o$, constants determined by the crystal's shape. (We return again to radii rather than diameters in order to match the above nomenclature.)

Equation (25) entails that a section of the crystal is approximated by a section of a paraboloid. Figure 11 depicts the radius r' of a section with the height $h_s$; the radius at the bottom of the section is $r'_0$, and on top of the section, $r'_1$.

Applying eq. (25) to $x = 0$ and $x = h_s$, respectively, permits to replace the constants k and $x_0$ with $r_0'$, $r_1'$, and $h_s$:

$$k = \frac{r_1'^2 - r_0'^2}{h} \qquad (26)$$

$$x_c = h_s \cdot \frac{r_0'^2}{r_1'^2 - r_0'^2} \qquad (27).$$

The volume V of the paraboloid section obtained from rotation of the shaded area in Fig. 11 around the x axis can be calculated as:

$$V = \pi \cdot \int_0^{h_s} r^2 \cdot dx \qquad (28)$$

With (25) through (28), we obtain finally:

$$V = \frac{\pi \cdot h_s}{2} \cdot (r_0'^2 + r_1'^2) \qquad (29)$$

During regular crystal growth, SHAPE accumulates the volume within one "slice" of the crystal by adding the volumes of "differential" cylinders with the diameter of the crystal calculated in the previous pass and with a height determined by the difference of the crystal length values for the current and the previous pass. This volume increment may be negative during meltback conditions, or if the new length value was less than the previous one due to noise superimposed on the seed and crucible position signals. A "slice" boundary is detected when the two byte integer representation of the crystal length exceeds a multiple of 64, which corresponds to a length difference of approximately 1.17 mm. A new "slice" is added to SHAPE's image of the crystal in memory which is kept in DIATAB, a 64 element floating-point array of squares of diameters, calculating the square of the diameter of a cylinder with a height equal to the distance from the previous slice boundary, and a volume equal to the sum of "differential" volumes accumulated since then. (Due to noise and the limited resolution of the crystal length, the height of this cylinder may be slightly greater than 64 length counts.) (A previous approach to approximate the crystal by slices of paraboloids proved to be unstable because errors of the previously calculated diameter squares propagated into the newly calculated data when eq. (29) was solved.) All entries in

DIATAB are shifted up one step, and the new diameter square is stored as the crystal's bottom diameter. The top entry is lost.

In order to prevent erratic diameter square average values from being entered in the table and from eventually corrupting the diameter calculation when the slice in question arrives at the encapsulant surface, the Fortran subroutine CHKDTB was provided which is part of the module DIACNT but called from SHAPE. CHKDTB compares the absolute value of the difference between the preceding and the current squares, and adjusts the new value to differ by not more than the specified limit (which is kept in the Variable XTLSHP) from its predecessor. This approach allows for greater absolute and relative diameter fluctuations in stages where the crystal diameter is small (e.g., in the early cone sections) and where such fluctuations are quite normal; it is more restrictive within the full-diameter crystal body. (The data stored in DIATAB is in square millimeters; XTLSHP must therefore be set to the maximum permitted difference between the squares of the diameters (in millimeters) of two adjoining crystal sections.)

During meltback conditions, i.e., when the crystal length decreases rather than increases with time, the entries in the array of diameter squares are shifted down one step when a slice boundary is reached; the top entry is reduplicated.

A subroutine of SHAPE, CALCSD, uses the entries in DIATAB to calculate the square of the diameter of the crystal at the surface of the boric oxide melt (corresponding to $r_1^2$ in our calculations). Since the position of the oxide melt surface relative to the crystal depends on the total height of the oxide melt, which is, in turn, a function of the total crystal volume immersed in the boric oxide, the melt height and the immersed volume must be re-calculated in each pass of CALCSD. The following procedure is used in CALCSD (compare Fig. 12):

The portion of the crystal immersed in the boric oxide melt is divided into slices of uniform height $h_s$ whose radii (or rather, diameter squares) are stored in DIATAB. The top and bottom slices are obviously exceptions to this rule. The bottom slice is the portion of the crystal grown since the last slice boundary was encountered; the height of the top slice is determined by the position of the oxide surface.

In order to determine $r_1^2$, CALCSD assumes that the encapsulant melt height did not change since the last pass. CALCSD first checks whether the boric oxide height is less than 75 millimeters, i.e., less than the length of the portion of the crystal whose diameter squares are stored in DIATAB. The oxide height is limited to the maximum permitted value, and an error output is triggered if this is not the case. In order to prevent the "Oxide Height Overflow" error from either being reported every ten seconds, or from eventually hiding any other error condition which is flagged by the same parameter of SHAPE, SHSTAT, a counter byte is incremented for every occurrence of this error; SHSTAT, in contrast, is set to indicate the problem only when the counter wraps around to zero after 256 increments. Depending on the number of iterations required in CALCSD, this may happen every 5 to 20 minutes if the condition persists continuously.

Having checked the oxide height, CALCSD subtracts the height of the bottom slice, $h_o$, from the previous melt height h and determines by a simple modulo operation the distance x the melt surface lies above the center of the last slice which is immersed to more than 50 percent of its height. The square of $r_1$ is obtained from a linear interpolation of the two adjacent entries in the table (in Fig. 12, from the squares of $r_4'$ and $r_5'$), i.e., by an interpolation which assumes a paraboloid shape of the current section. Using the centers of the slices rather than the slice boundaries as top and bottom surfaces of a paraboloid section guarantees a better accuracy.

The melt height h, however, may have changed since the previous pass if the volume added at the growth zone was not equal to the volume withdrawn from the boric oxide. In order to determine the current value of h according to eq. (7), the immersed crystal volume $V_i$ must be known. For a crystal with n slices covered by the oxide melt to at least 50 percent of their height, we can calculate $V_i$ according to:

$$V_i = V' + \pi \cdot h_s \cdot [r_0'^2 + r_1'^2 + \ldots + r_{n-1}'^2 + r_n'^2/2] + \pi \cdot x \cdot (r_n'^2 + r_1^2)/2 \qquad (30).$$

(V' is the volume of the currently grown section of the crystal with the height $h_o$.)

Equation (7) permits now to calculate a new melt height value (assuming the oxide volume $V_m$ and the crucible radius R are known) which is compared to the previous height. CALCSD returns if both values differed for less than one height unit (approximately 0.02 mm); otherwise, the procedure is repeated from the calculation of $r_1$ on. CALCSD is left, though, if a certain number of iterations (currently, 5) was not sufficient, which constitutes a protection against "hanging up" in the case of bad convergence.

During the major part of the growth run, the oxide volume $V_m$ in eq. (7) is, indeed, known and constant. Towards the end of the run, however, the active boric oxide volume starts decreasing as the semiconductor melt recedes from the crucible wall, and the resulting gap is filled with boric oxide. Therefore, $V_m$ has to be corrected after each pass in this regime for the apparent oxide volume loss $\delta V_m$ according to eq. (9):

$$V_m = V_{mo} + \Sigma\, \delta V_m = V_{mo} - \Sigma\, (\epsilon \cdot \delta V_2 . d_x / d_{xm}) \qquad (31),$$

where $V_{mo}$ is the initial boric oxide melt volume.

One more task of SHAPE is the evaluation of a Crucible Position setpoint which also enters into the calculation of the Length Grown. The apparent weight of the growing crystal, W, can be written as

$$W = W_o + W_x - (V_i - V_{io}) \cdot d_o \qquad (32),$$

where $W_o$ is the measured initial weight at the beginning of the growth run, and $W_x$, the actual weight of the crystal grown. The last term in eq. (32) takes into account the buoyancy in the boric oxide melt. The mass $W_x$ has been withdrawn from the contents of the crucible, essentially by lowering the surface of the semiconductor melt. Towards the end of the growth run, however, the semiconductor melt volume required to grow the crystal is supplied by shrinking the diameter rather than the height of the semiconductor melt. The volume thus obtained is identical to the boric oxide volume lost according to eqs. (9) and (31). The crucible must be raised by a distance z in order to keep the semiconductor melt surface at the same location within the puller:

$$W_x = R^2 \cdot \pi \cdot z \cdot d_{xm} - \Sigma\, \delta V_m \cdot d_{xm} \qquad (33)$$

(Keep in mind that $\delta V_m$ is negative; the contribution of the right term in eq. (33) is therefore either zero - during the regular growth - or positive.)

The density of the melt, $d_{xm}$, is different from the density of the crystal, $d_x$. The crucible position setpoint, $z_s$, can be calculated from eqs. (32) and (33) with the initial crucible position $z_o$:

$$z_s = z_o + z =$$

$$= z_o + \frac{W + V_i \cdot d_o - (W_o + V_{io} \cdot d_o) + \Sigma\, \delta V_m \cdot d_{xm}}{R^2 \cdot \pi \cdot d_{xm}} \qquad (34)$$

We may substitute with eqs. (7) and (31) for the immersed volumes $V_i$ and $V_{io}$, and we obtain with the melt height $h_o$ at the beginning of the growth run:

$$z_s = \frac{W}{R^2 \cdot \pi \cdot d_{xm}} + h \cdot \frac{d_o}{d_{xm}} + \frac{\Sigma\, \delta V_m}{R^2 \cdot \pi} \cdot \left(1 - \frac{d_o}{d_{xm}}\right) +$$

$$+ \left(z_o - h_o \cdot \frac{d_o}{d_{xm}} - \frac{W_o}{R^2 \cdot \pi \cdot d_{xm}}\right) \qquad (35)$$

The term in the second line of eq. (35) is an initialization constant. The calculated Crucible Position setpoint is returned by SHAPE in the INTEGER*2 Variable SCRUCP.

The actual position $z_a$ of the crucible as obtained from the Crucible Position potentiometer may be different from $z_s$; with the initial and actual seed position values $x_o$ and $x_a$, respectively, the length l of the crystal can be calculated as:

$$l = (x_a - x_o) + (z_s - z_a) \qquad (36).$$

An accordingly determined Crystal Length value is returned by SHAPE in the Variable ILENGT (in INTEGER*2 notation); it is eventually converted to floating-point format and scaled by the Command Executor and stored as LENGTH.

Note that different approaches are used for the calculation of the actual Growth Rate and of the Crucible Position setpoint and Length Grown values. In the first case, the Growth rate is derived from (measured) <u>speed</u> values, while the Crucible Position setpoint calculation is based on the <u>weight</u> of the crystal. Although this approach may appear redundant, it was indispensable in order to obtain an acceptable accuracy of the results. (In general, it is preferable to use an input value which constitutes already an integral magnitude (such as the crystal weight), rather than calculating the integral; similar considerations apply to derivative data such as speeds.)

A Meltback condition is indicated by SHAPE and subsequently reported by DIACNT if the Crystal Length value calculated was decreased by more than one "slice", i.e., by more than 1 mm. This may be due to any effect which reduces the distance between the seed and the semiconductor melt surface, whether it was caused by a movement of the seed, or of the crucible. A "Regular growth resumed" message is similarly issued after a RESET command, upon the detection of a non-zero Seed Lift speed after a "Zero seed lift speed" error, or if the Crystal

Length was increased again by more than one "slice" after a Meltback condition.

SHAPE is called as an INTEGER*1 Function from Fortran; the Differential Weight is passed to it as a parameter. It returns an integer flag which indicates the status of its operation. The following values are currently defined:

| | |
|---|---|
| 3 ... | Oxide Height overflow. |
| 2 ... | Seed Lift speed is zero - no diameter calculated. |
| 1 ... | Meltback. |
| 0 ... | Regular growth. |
| -1 ... | Speed overflow - RESET or REACTV call required. |
| -2 ... | SHAPE is not yet initialized - RESET required. |

All other parameters are passed to and from SHAPE via memory locations in COMMON blocks most of which can be accessed as Variables.

### 4.5.2.4 THE INITIALIZATION OF THE ROUTINE SHAPE - ROUTINE RESET

Although the subroutine RESET is logically part of the Command Executor Task CMMDEX, it is kept in one assembly language module together with SHAPE, and it is discussed here. The essential commission of RESET is to prepare SHAPE for its operations. RESET has to be called before usable Diameter, Length grown, and Crucible Position setpoint values can be obtained from SHAPE if the status value returned by SHAPE is negative (compare chapter 4.5.2.3). This value is negative

(a) After the start of the system, before RESET was called the first time, and

(b) After a "Speed overflow" error which happens if SHAPE is no more able to update its stack of crystal "slices". This is the case if less than one diameter value per "slice" is available, corresponding to speeds in excess of 400 mm/h.

(In the latter case, a call to the subroutine REACTV may be sufficient to re-establish proper operation of SHAPE; compare chapter 4.5.2.5.)

The following items are initialized by RESET:

(1) The initial values of the crystal Weight, the Seed, and the Crucible Positions.

(2) The Diameter Square Table DIATAB is filled assuming a cylindrical crystal with a diameter equal to the Seed Diameter which was specified with the INITIALIZE function. The initial immersed crystal volume is calculated accordingly.

(3) The initial melt height is derived from the Melt Weight and Density values obtained from INITIALIZE, and from the above initial immersed volume.

(4) The parameter ALPHA (compare chapter 4.5.2.3) is set to 1, and the encapsulant volume "lost" by melt recession (compare eq. (9)) is reset to zero.

### 4.5.2.5 THE RE-ACTIVATION OF SHAPE - ROUTINE REACTV

The subroutine REACTV is kept in one module together with SHAPE. It permits to safely resume the operation of SHAPE after a Speed Overflow error which was not likely to have caused significant changes to the volume of the crystal immersed in the boric oxide melt. REACTV simply resets the internal location which is used to accumulate the volume of the crystal "slice" grown since the last pass of SHAPE, and it activates SHAPE again by resetting its status byte.

## 4.5.3 THE ANALOG DATA CONTROLLER - TASK ANACNT

### 4.5.3.1 THE ANALOG CONTROLLER ROUTINE PROPER - MODULE ANACNT

The main part of the Analog Data Controller Task is kept in the Fortran module ANACNT. The following operations are done by the Analog Data Controller Task:

(1) ANACNT requests from the A/D Converter board the A/D converted values for 25 analog input channels, and it preprocesses the Weight value by subtracting the offset weight determined by RESET.

(2) It controls the power output for three heaters, running PID controller routines with the Heater Temperature setpoints and actual values as inputs.

(3) It generates similarly the control output to the four motors.

(4) It provides input from and output to the Motor Direction relay circuitry, and it takes care of the Controller Selection output.

(5) It writes the control data determined above and the Plot data collected by the Command Executor (compare chapter 4.4.4.7) to the D/A Converter hardware.

ANACNT runs once every second, being triggered directly by the Timer Task. Immediately after having been started, it sets an auxiliary seconds flag, SECFLG, which, in turn, sets to work the Command Executor.

The first major operation of ANACNT is obtaining input data from the A/D Converter board. This is done within the assembly language subroutine ANAINP (compare chapter 4.5.3.2). The ANAINP call is skipped if the flag TEST is set to -1. ANAINP returns the input from the Converter hardware in the 25 element array ANALOG, as two-byte integer data. (The contents of this array may be patched with arbitrary simulation data which can even be ramped if required if ANAINP was disabled with TEST.)

Having read the input data, ANACNT prepares the data and parameter locations for the seven PID controllers which are run by this task. The three Temperature controllers (one for each zone of a three-zone heater) use the pertinent Heater Temperature setpoints and actual values as input data; the Power Limit setpoint serves as a common Limit value for all three controllers. The controllers' Bias values are kept at zero.

A totally different approach is used for the Motor Speed controllers: On principle, the motor controller hardware could be driven directly by the D/A converted speed setpoints. Due to nonlinearities and offset errors within the motor controller hardware, this approach would result in unsightly differences between the speed setpoints and the actual speeds. In order to alleviate this problem, one PID controller was provided for each of the four motor channels which receives the pertinent setpoint as an input and as a Bias value and which generates output data which is eventually fed to the motor controller hardware, using the actual motor speed as its second input. In contrast to the Temperature controllers which employ genuine PID operation, the Motor controllers are currently programmed to operate as integral controllers only; their only task is to correct the original setpoint values slightly in order to force the difference between a Motor Speed setpoint and the pertinent actual value to zero.

ANACNT prepares the inputs to the PID routines in any case; it runs the PID controllers only if the CGCS is in charge of the puller. In contrast, it resets the Previous Error and Error Integral locations of all controllers while the system is in Monitoring mode, and it provides the Motor Speed setpoints for output by the D/A Converter board. (This provision was made in order to permit an easier test of the output hardware. The Motor Speed control signals are thus available at the outputs of the Digital Controller in any case.)

A special treatment is required for the Motor Speeds: Due to the offset usually introduced by the PID controllers, a non-zero output signal results even if the corresponding setpoint was actually set to zero. Although this non-zero output signal might only compensate for an opposite offset of the hardware, it would prevent the Motor Direction Relay controller routine MOTDIR (compare chapter 4.5.3.3) from switching the motors actually off. ANACNT branches therefore according to the Motor Speed setpoint values, and provides a zero output explicitly when required.

Having trapped possibly negative Temperature Controller output values (there is no negative heater power), ANACNT copies the internal array of analog input data, ANALOG, to the array ANADAT, shifting the contents of the ANALOG array by one element. This was done to guarantee that the important input data (i.e., the first 17 elements of ANADAT) is actually sampled at the same time. The first element in ANALOG was measured at the end of the previous call to the Analog Data Input routine ANAINP (compare chapter 4.5.3.2), and it is therefore approximately one second older.

Finally, ANACNT calls the Motor Direction Relay controller routine MOTDIR (compare chapter 4.5.3.3), and writes the fifteen Analog Output values in the array ANAOUT (three temperatures, four motors, and eight chart recorder output channels) to the D/A Converter board, calling the Analog Data Output routine ANAOPT (compare chapter 4.5.3.4). Both operations are skipped if TEST is set to -1.

### 4.5.3.2 THE ANALOG DATA INPUT ROUTINE ANAINP

ANAINP is an assembly language routine which reads data from the A/D Converter board in a random access mode, and submits the values obtained to digital low-pass filtering.

In order to permit random input of data from the hardware, ANAINP uses a special parameter array ANIPAR which consists of two bytes for each channel. It is, therefore, very easy to connect a logical data channel within the CGCS to an arbitrary hardware channel and to modify the gain and filtering parameters of any channel by changing the contents of ANIPAR. In addition, the number of input channels actually read is not built into ANAINP but derived from the parameter array ANIPAR: The operation of ANAINP is terminated, and the routine returns to the calling task, when the most significant bit of an odd element of ANIPAR is set, corresponding to any negative value. (The remainder of the parameter byte does not matter.) It is therefore essential that at least one negative value is provided in ANIPAR lest ANAINP might indefinitely continue reading data; since the output is stored in an array which is specified as the second parameter of ANAINP, this data input would exceed the boundaries of the array and eventually overwrite important data. The two parameter bytes per channel in ANIPAR hold the following information:

<u>ANIPAR (2n + 1):</u> (n = 0, 1, 2, 3 ...)

```
Bit   7    6    5    4    3    2    1    0
      |    |____|    |__________________|
      |    |         |
      |    |        A/D Converter Channel (0 ... 31)
      |   Gain:
      |    0    0  =  1
      |    0    1  =  2
      |    1    0  =  4
      |    1    1  =  8
      |
      0 ... Active input channel
      1 ... Last entry in ANIPAR; bits 0 to 6 don't matter
```

<u>ANIPAR (2n + 2):</u> (n = 0, 1, 2, 3 ...)

Low-pass filter flag, determines the cut-off frequency of the digital low-pass filter routine:

| Value | Cut-Off Frequency (Hz) |
|---|---|
| 0 | infinite |
| 1 | 0.1150 |
| 2 | 0.0461 |
| 3 | 0.0213 |
| 4 | 0.0103 |

ANAINP supposes that a valid result is held by the A/D Converter hardware for the first input channel. Correspondingly, the last action of ANAINP prior to its return, and one of the actions of the initialization routine for ANAINP, ANAINI, is to prepare and trigger the conversion of the first input channel. Since approximately one second passes between the return from ANAINP and the next call to this routine, this data is already slightly outdated when it is retrieved at the beginning of the next pass of ANAINP, which may or may not matter. Each input value is immediately submitted to the digital low-pass filter routine in LOWPAS which corresponds to a first order analog low-pass (compare chapter 4.5.3.5). LOWPAS needs the previous data value of each analog channel which it deposited in the output array ANALOG; the contents of this array may, therefore, only be read but not modified. (This is not true if ANAINP which calls LOWPAS is disabled altogether with the TEST flag.) Prior to calling LOWPAS, ANAINP prepares the A/D Converter board for the input of the next channel, programming the input multiplexer accordingly. The set-up time required by the multiplexer, i.e., the time which must pass before valid data can be submitted to the A/D Converter proper, is approximately equal to the execution time of LOWPAS. The A/D Converter hardware will therefore be ready for the next step when LOWPAS has finished its job. ANAINP triggers the conversion proper when the A/D Converter board's hardware indicates that the board is ready; the routine waits in a loop until converted data is available. (The synchronization with the hardware is done with waiting loops rather than using interrupts. This approach was preferable because each interrupt processed involves a considerable system overhead which takes several hundred microseconds. The A/D conversion is even faster than the processing of an interrupt, and the time required by the hardware for channel switching is used within ANAINP for the low-pass routine call.) Emergency timeouts were provided for either loop in order to avoid a total blockage of the system if the A/D Converter does not respond properly. (It turned out that the system _is_ blocked, though, if no A/D Converter board is installed, and ANAINP is _not_ disabled with TEST.)

### 4.5.3.3 THE RELAY CONTROLLER ROUTINE MOTDIR

This assembly language routine provides the input from and the output to the digital (relay) interface. It has the following tasks:

(1) Provide output to the Controller Selection relay which must not be energized in Monitoring mode 0, and energized if the CGCS is in charge of the puller (i.e., in operation modes 1 through 4).

(2) Read the current status of the Motor Direction relays, and

set the Motor Speed input values to zero if the corresponding motor is switched off.

(3) Check the sign of the Motor Speed output values, and provide the proper Motor Direction relay output.

(4) Determine the absolute value of the Motor Speed output for the D/A Converter.

A special approach had to be chosen for the relay output: Using just one output bit for turning on and off one relay would have been impeded by the fact that the status of the output ports may be undefined when the CGCS is not in charge of the controller computer. Furthermore, the PPI (Peripheral Parallel Interface) used comes up with all I/O lines in high impedance after a system reset, which would result in all relays either turned on or off. Therefore, three bits, namely bits 0 through 2 of one output byte, have to be set to defined values in order to actually permit control of the relays: Bit 0 represents the Controller Selection output; it has to be high to switch control to the CGCS and to activate the Motor Direction relays. In addition, bit 1 must be low, and bit 2, high, to enable the relays.

The Cambridge Motor Controller uses three relays for Motor Up/Clockwise, Motor Stop, and Motor Down/Counterclockwise, respectively. The CGCS is therefore connected to the puller by three relay control lines for each motor; these lines are energized by the Cambridge console if the analog circuitry is controlling the puller, and by the CGCS, if the CGCS is in charge. The status of the relay control lines is monitored by MOTDIR, and MOTDIR provides output to them when required. Since exactly one of the Motor Control relays must be energized for each channel, the status of the three lines may be represented by two bits:

| Output: | Motor Status: | Speed Value: | Input: |
|---|---|---|---|
| 0 0 | Stop | 0 | 0 0 |
| 1 0 | Up/Clockwise | + | 1 0 |
| 0 1 | Down/Counterclockwise | - | 0 1 |
| 1 1 | Stop | 0 | 0 0 |

MOTDIR uses two relays for the Motor Direction output whose contacts are wired to result in the above signals, i.e., the Stop line is energized if either no relay or both of them are on.

First, MOTDIR reads the current Motor Direction status, and resets the Motor Speed input value to zero if the Stop relay is on. This step prevents noise and offset errors within the analog circuitry from disturbing the Motor Speed output on the CGCS's console. The four times two Direction input bits are read and internally stored as one byte. Next, MOTDIR checks the Motor Speed setpoint values, and determines a relay setting according to the magnitude and sign of each setpoint. These four times two bits are also assembled in one byte. The input and output bytes are now "or"-ed, which sets all bits in the output byte which are set in one of the two input bytes, or in both. The two bits corresponding to one particular motor are reset to zero if a zero Speed value was submitted as a setpoint. This resulting byte is output to the Motor Direction relays in any case. Actual output to the control lines is, however, only generated if the CGCS is in charge of the puller.

The chosen approach may appear unnecessarily complicated but it is, in fact, indispensable to guarantee valid Motor Direction signals. The combination of the output data with the previous input data has no effect if the direction of the setpoint speed is the same as the actual speed; the current motor direction will be maintained. The puller requires, however, a few fractions of a second in "Motor Stop" position if the rotation direction of a motor is to be reversed. This is automatically accomplished by the chosen approach: Both bits corresponding to one motor are set if the actual Motor Speed and the pertinent setpoint have different signs, which energizes the "Stop" control line. At the next pass of MOTDIR, one second later, two zero bits are input accordingly, and the Motor Direction output will be determined by the sign of the setpoint. A one second "Motor Stop" is therefore guaranteed in any case.

### 4.5.3.4 THE ANALOG DATA OUTPUT ROUTINE ANAOPT

The Analog Data Output routine uses a similar approach as ANAINP for providing easily programmable output to random hardware channels of the D/A Converter board: The channel numbers are kept in an array whose size is not limited by ANAOPT. Output values are read from an array in the order in which they are stored; there is no limit to this array either. ANAOPT returns to the calling routine when a negative channel number is detected in the parameter array (which is referred to as the Variable ANOPAR).

ANAOPT has to scale the data submitted to it by a factor of 8 since the D/A Converter board supports only a 12 bit unipolar data range. Round-off is provided according to the magnitude of the highest-order bit which has to be discarded. Negative output values are trapped and replaced by zero.

Incidentally, the (assembly language) routine ANAOPT is kept in the Data rather than in the Code area of the CGCS. This was necessary because ANAOPT "patches" its own program code according to the analog channel which is currently in use. This approach is, however, incompatible with the memory checking done by the Command Executor (compare chapter 4.4.4.8). (Although patching program code is legitimately considered a bad programming technique it was indispensable in this case because the 8085 processor used does not allow otherwise to access I/O port addresses which have been calculated before.)

### 4.5.3.5 THE LOW-PASS FILTER ROUTINE LOWPAS

The algorithm used by LOWPAS is very simple and efficient: With $x_k$, the current input value, and $y_k$ and $y_{k-1}$, the current and the previous output values, respectively, LOWPAS calculates:

$$y_k = a \cdot x_k + b \cdot y_{k-1} \qquad (1),$$

with

$$a = 2^{-n} \qquad (2),$$

and

$$b = 1 - a = 1 - 2^{-n} \qquad (3),$$

where k and n are positive integers (0, 1, 2, ...). The restriction of eq. (2) permits a very fast evaluation of eq. (1). Eq. (3) guarantees an overall gain of 1 for constant (DC) signals. We can re-write eq. (1) to:

$$b \cdot (y_k - y_{k-1}) + (1 - b) \cdot y_k = a \cdot x_k \qquad (4)$$

Eq. (3) can be divided by T, the time interval between two runs of LOWPAS:

$$\frac{y_k - y_{k-1}}{T} + \frac{1 - b}{b \cdot T} \cdot y_k = \frac{a}{b \cdot T} . x_k \qquad (5)$$

The difference in the left term in eq. (5) can be approximated by a differential, transforming eq. (5) to the differential equation:

$$\frac{\delta y}{\delta t} + \frac{1 - b}{b \cdot T} \cdot y = \frac{a}{b \cdot T} \cdot x \qquad (6)$$

This is evidently the response of a simple first-order (R-C) low-pass filter. Eq. (5) is an approximation, though, which is only valid for very slowly changing input values $x_k$. A more accurate analysis of the filter's frequency response must be based on the theory of digital filters: The complex frequency response $H(\Omega \cdot T)$, with

$$\Omega = 2 \cdot \pi \cdot f \qquad (7),$$

where f is the input signal frequency and T the time interval between two sampling points, can be obtained by a z-transformation of the filter's response to a single pulse with the amplitude 1. It can easily be seen from eq. (1) that, for an input signal

$$\delta_k = \begin{cases} 1 & \text{for } k = 0 \\ 0 & \text{for } k > 0 \end{cases} \qquad (8),$$

the output signal $h_k$ will be:

$$\begin{array}{lccccccc} k & = & 0 & 1 & 2 & 3 & & m \\ h_k & = & a & a \cdot b & a \cdot b^2 & a \cdot b^3 & \ldots & a \cdot b^m \end{array} \qquad (9)$$

With the definition of $H(\Omega \cdot T)$

$$H(\Omega \cdot T) = \sum_{k=0}^{\infty} h_k \cdot [\exp\,(j \cdot \Omega \cdot T)]^{-k} \qquad (10),$$

and the summation formula for an infinite geometric series

$$1 + x + x^2 + x^3 + \ldots = \frac{1}{1 - x} \qquad (11),$$

we can obtain the complex frequency response

$$H(\Omega \cdot T) = \frac{a}{1 - b \cdot \exp\,(-j \cdot \Omega \cdot T)} \qquad (12).$$

Since we are not interested in the phase but only in the amplitude response, we derive the absolute value $|H(\Omega \cdot T)|$ from eq. (12):

$$|H(\Omega \cdot T)| = \frac{a}{(1 + b^2 - 2 \cdot b \cdot \cos\,(\Omega \cdot T))^{1/2}} \qquad (13)$$

The cut-off frequency

$$\Omega_o = 2 \cdot \pi \cdot f_o \tag{14}$$

of a low-pass filter is defined as the point where the amplitude response drops to $1/\sqrt{2}$ of its DC value:

$$\frac{|H(\Omega_o \cdot T)|}{|H(0)|} = \frac{1}{\sqrt{2}} \tag{15}$$

With eqs. (13) and (15), we can write:

$$\cos\,(\Omega_o \cdot T) = 1 - \frac{(1 - b)^2}{2 \cdot b} \tag{16}$$

The cut-off frequency values listed in chapter 4.5.3.2 were obtained from eq. (16), with a sampling point interval T = 1 second.

## 4.6 PROGRAM CONFIGURATION

The CGCS consists of a number of Fortran and assembly language program source modules each of which holds one or several routines. These modules must first be converted into object machine code, which is done by a Compiler and Assembler program, respectively. The resulting object program files must be linked together by a special Linker utility which also resolves mutual references; the output of the Linker which still does not refer to absolute memory locations must be modified to do so by a Locater. A special Configuration Module must be provided for the iRMX-80 operating system; this module may either be written in assembly language, or it can be created much more comfortably with a special Interactive Configuration Utility for iRMX-80, ICU-80. (All the mentioned development software is supplied by Intel.)

The actual configuration process is, however, much more complicated, due to the overlay structure chosen, and due to the fact that certain memory locations have to be "tied" together. In general, the configuration procedure follows the approach described in the Fortran-RMX-80 Interface documentation (compare Appendix A), particularly with the treatment of COMMON blocks which we will not discuss here due to its complexity.

The configuration procedure starts with linking all assembly language and Fortran modules together which constitute the main body of the CGCS (i.e., the permanently resident code). These routines refer extensively to Interface, Fortran, and iRMX-80 routines which are linked with the combined assembly language and Fortran modules in the next step. A Dummy module, TRVMOD, provides references which would be made by overlay routines otherwise; this permits to keep all support routines within the resident code. After this procedure, all references to external routines should be satisfied, with the exception of the routines which constitute Command Interpreter overlays. Despite of these missing external references, the resident code is located to absolute memory addresses.

Next, the overlay routines (which may be one or more program modules per overlay) are linked separately, satisfying their external references to routines which are already contained in the resident part of the CGCS. (The addresses of these routines only are linked in this case, rather than the complete program code.) Since no overlay may directly refer to a different overlay, all references contained in the overlay must be satisfied now, and the overlay code can be located to reside in the reserved overlay area.

The last step, finally, entails linking the resident CGCS code to the start addresses of all Command Interpreter overlays, which satisfies the last yet open external references. In addition, the Initialization code of the Command Interpreter which was prepared separately like an overlay is linked to the resident CGCS code in its entirety. The resulting modules still contain a vast overhead of various references which were required for linking and which are used by various debugging approaches. These references are not required for program execution and would only unduly consume disk space and loading time; they are, therefore, stripped in the last step of software preparation.

In addition to the main resident CGCS module CZOCHR.BIN and the 21 Command Interpreter overlay modules CZOV01 through CZOV21, plus the Data module CZOOVD, two more files are required on a CGCS system disk: The file CZOMEN holds a specially formatted Help menu which is displayed upon a HELP command (compare chapter 4.4.3.4); this file needs no special attention if a new system version is being generated, unless significant modifications of the command structure were made. A special treatment is, in contrast, required for the CZONAM file which holds the list of Variable addresses (compare chapters 3.6 and 4.4.3.2). This file must hold the current version code in its file name extension (CZONAM.V23 refers, for example, to version 2.3 of the CGCS), and it must be generated from a source file which may have required updating due to a possible shift of the addresses of some Variables because of software modifications. This source file is converted into the special CZONAM format (compare Appendix H) by means of the auxiliary program CONVAD. CONVAD is designed to run under ISIS-II; it could also be executed under RXISIS-II but that will hardly be necessary.

The last step in preparing a work disk for a new CGCS version is, finally, up to the operator: All Macro command files which were used under a previous version and which are still required must be converted to the new system version using the Macro Command Editor facilities (compare Appendix A).

## APPENDIX A: ADDITIONAL DOCUMENTATION

iRMX-80$^{TM}$ User's Guide; Intel Corporation 1979, 1980; Manual Order No 9800522-05:
General information about iRMX-80.

The Alternative Loader Task - Library RQLOAD.LIB; Karl Riedling, September 1984:
Information within this documentation complements and replaces information in the iRMX-80$^{TM}$ User's Guide.

The Alternative Terminal Handler - Library ATHxxx.LIB; Karl Riedling, February 1985:
Information within this documentation complements and replaces information in the iRMX-80$^{TM}$ User's Guide.

RXISIS-II User's Guide; Karl Riedling, February 1985:
General information about RXISIS-II and its supporting routines, the RXISIS-II Monitor, and the RXISIS-II Confidence Test. Short overview over utility software available under RXISIS-II.

ISIS-II User's Guide; Intel Corporation, various issues and order numbers:
Documentation of Intel supplied utility software which is also available under RXISIS-II.

Additional System Programs for Intel Development Systems; Karl Riedling, March 1981:
Documentation of additional utility routines; all programs listed are compatible with RXISIS-II.

Fortran - RMX-80 Interface Program Package; Karl Riedling, February 1985 (Issue 3):
Extensive documentation of all Interface routines used in the CGCS, containing also discussions of various programming approaches and of the system configuration.

Additional Fortran Numeric Routines; Karl Riedling, 1985:
Documentation of alternative Fortran floating-point system routines which use the 8231 Numeric Processor.

Czochralski GaAs Crystal Growth Controller - Short Reference; Karl Riedling, December 1986 (Issue 4):
User's reference manual for the CGCS.

Czochralski GaAs Crystal Growth Controller - Operator's Manual; Karl Riedling, December 1986:
Operation guide for RXISIS-II and the CGCS. (Subset of the Short Reference Manual.)

<u>Czochralski Growth Control System - Digital Controller Emergency Procedures;</u> Karl Riedling, December 1986:
Procedures for emergencies caused by the CGCS hardware or software. (Subset of the Operator's Manual.)

<u>Czochralski Growth Control System Macro Command Editor COMMED;</u> Karl Riedling, April 1986:
User's reference manual for the Macro Command Editor programs COMMED and READCM.

<u>Program SHODAT - Short Reference;</u> Karl Riedling, May 1986 (Issue 2):
User's reference manual for the Data file display utility SHODAT.

# APPENDIX B: CGCS MEMORY AND I/O MAPS

## B 1: MEMORY MAP

| | |
|---|---|
| FFFFH | |
| | Loader Buffer, Disk I/O Stack |
| FEB0H | |
| | System Version Code (2x) |
| FEACH | |
| | RXIROM (COMINT) Stack |
| FE34H | |
| | Disk Buffer Area |
| FD30H | |
| | Memory Pool Area |
| ≈F5F0H * | |
| | Resident CGCS Program Code |
| 5C00H | |
| | COMINT Overlay Program Code + Data |
| 5400H | |
| | Resident CGCS Data |
| 2D00H | |
| | COMMON Blocks |
| 2800H | |
| | Data of ROM Resident System |
| 2000H | |
| | ROM Resident Program Code |
| 0000H | |

* This boundary is most subject to changes due to program modifications. The value given applies to Version 2.3.

## B 2: I/O MAP

| | | |
|---|---|---|
| 20H | ... | A/D Converter Control/Status Register, low byte |
| 21H | ... | A/D Converter Control/Status Register, high byte |
| 24H | ... | A/D Converter Multiplexer Address Register |
| 26H | ... | A/D Converter Output Data Register, low byte |
| 27H | ... | A/D Converter Output Data Register, high byte |
| 40H | ... | D/A Converter Channel 0, low byte |
| 41H | ... | D/A Converter Channel 0, high byte |
| 42H | ... | D/A Converter Channel 1, low byte |
| 43H | ... | D/A Converter Channel 1, high byte |
| ... | | |
| 5EH | ... | D/A Converter Channel 15, low byte |
| 5FH | ... | D/A Converter Channel 15, high byte |

B0H ... I/O Expansion Board Base Address

B4H ... Motor Direction Relay Input
B5H ... Motor Direction Relay Output
B6H ... Controller Selection Relay Output

C0H - FFH ... CPU Board I/O Addresses

Various I/O ports on the iSBC 80-24 CPU and iSBC 517 I/O Expansion boards are used by system routines, e.g., by the Terminal Handler and the alternative Fortran floating-point routines.

## APPENDIX C:  SYSTEM TASKS

Appendix C lists all primary tasks within the system; it does not include tasks which are dynamically created at runtime by any primary task.  SUSPEND information is given for the genuine CGCS tasks; it indicates whether a task may be suspended with the DEBUG Suspend command.  It is generally prohibited to suspend any iRMX-80 System or Interface task!

### C 1:  ROM RESIDENT SYSTEM TASKS

Task RXIROM:  ROM resident root of RXISIS-II and the CGCS Command Interpreter.

| | |
|---|---|
| Entry Point: | RXIROM |
| Stack Length: | 50, extended to 120 by the CGCS |
| Priority: | 250 |
| Task Descriptor: | RXIRTD |
| Extra: | 20 |

Task RQTHDI:  Alternative Input Terminal Handler.

| | |
|---|---|
| Entry Point: | RQTHDI |
| Stack Length: | 40 |
| Priority: | 97 |
| Task Descriptor: | THDITD |
| Extra: | 0 |

Task RQTHDO:  Alternative Output Terminal Handler.

| | |
|---|---|
| Entry Point: | RQTHDO |
| Stack Length: | 40 |
| Priority: | 113 |
| Task Descriptor: | THDOTD |
| Extra: | 0 |

Task RQLOAD:  Alternative Loader Task.

| | |
|---|---|
| Entry Point: | RQLOAD |
| Stack Length: | 60 |
| Priority: | 140 |
| Task Descriptor: | LOADTD |
| Extra: | 0 |

Task DISKIO: iRMX-80 Disk I/O Task.

| | |
|---|---|
| Entry Point: | RQPDSK |
| Stack Length: | 48 |
| Priority: | 129 |
| Task Descriptor: | RQDIOD |
| Extra: | 0 |

Task : Unnamed Disk Controller Task.

| | |
|---|---|
| Entry Point: | RQHD4 |
| Stack Length: | 80 |
| Priority: | 33 |
| Task Descriptor: | CNTLTD |
| Extra: | 0 |

## C 2: iRMX-80 SYSTEM TASKS IN THE CGCS

Task RQFMGR: Free Space Manager.

| | |
|---|---|
| Entry Point: | RQFMGR |
| Stack Length: | 40 |
| Priority: | 50 |
| Task Descriptor: | RQFSMD |
| Extra: | 0 |

Task DIRSVC: Disk Directory Services.

| | |
|---|---|
| Entry Point: | RQPDIR |
| Stack Length: | 48 |
| Priority: | 200 |
| Task Descriptor: | RQDRSD |
| Extra: | 0 |

## C 3: FORTRAN - iRMX-80 INTERFACE TASKS

Task FXCFLG: Flag Interrupt Generation Task.

| | |
|---|---|
| Entry Point: | FXCFLG |
| Stack Length: | 36 |
| Priority: | 149 |
| Task Descriptor: | FXCFTD |
| Extra: | 0 |

Task INDATX: Input Interface Task.

| | |
|---|---|
| Entry Point: | FXINTI |
| Stack Length: | 184 |
| Priority: | 134 |
| Task Descriptor: | INDTTD |
| Extra: | 18 |

Task OUTDTX: Output Interface Task.

| | |
|---|---|
| Entry Point: | FXINTO |
| Stack Length: | 200 |
| Priority: | 135 |
| Task Descriptor: | OUTDTD |
| Extra: | 18 |

Task FXDISK: Disk I/O Interface Task.

| | |
|---|---|
| Entry Point: | FXDISK |
| Stack Length: | 38 |
| Priority: | 133 |
| Task Descriptor: | DISKTD |
| Extra: | 0 |

Task FXTIME: System Timer Task.

| | |
|---|---|
| Entry Point: | FXTIME |
| Stack Length: | 34 |
| Priority: | 34 |
| Task Descriptor: | TIMETD |
| Extra: | 0 |

## C 4: CONTROLLER TASKS

Task CMMDEX: Command Executor Task.

| | |
|---|---|
| Entry Point: | CMMDEX |
| Stack Length: | 120 |
| Priority: | 240 |
| Task Descriptor: | CMEXTD |
| Extra: | 20 |
| Suspend: | no |

Task MEASDO: Measured Data Output Task.

| | |
|---|---|
| Entry Point: | MEASDO |
| Stack Length: | 120 |
| Priority: | 220 |
| Task Descriptor: | MEASTD |
| Extra: | 20 |
| Suspend: | yes |

Task CMFINP: Command File Input Task.

| | |
|---|---|
| Entry Point: | CMFINP |
| Stack Length: | 50 |
| Priority: | 230 |
| Task Descriptor: | CMFITD |
| Extra: | 20 |
| Suspend: | yes |

Task CMFOUT: Command File Output Task.

| | |
|---|---|
| Entry Point: | CMFOUT |
| Stack Length: | 50 |
| Priority: | 251 |
| Task Descriptor: | CMFOTD |
| Extra: | 20 |
| Suspend: | yes; for short time only |

Task DSKOUT: Data Disk File Output Task.

| | |
|---|---|
| Entry Point: | DSKOUT |
| Stack Length: | 50 |
| Priority: | 180 |
| Task Descriptor: | DSKOTD |
| Extra: | 20 |
| Suspend: | yes |

Task DIACNT: Diameter Controller Task.

| | |
|---|---|
| Entry Poi[illegible] | DIACNT |
| Stack Len[illegible] | 120 |
| Priority: | 160 |
| Task Descriptor: | DIACTD |
| Extra: | 20 |
| Suspend: | yes |

Task ANACNT: Analog Data Controller Task.

| | |
|---|---|
| Entry Point: | ANACNT |
| Stack Length: | 60 |
| Priority: | 150 |
| Task Descriptor: | ANACTD |
| Extra: | 0 |
| Suspend: | no |

## APPENDIX D: ROUTINE NAMES

The following table lists the names of all routines which do not belong to iRMX-80 or Interface libraries. The name of the source file which holds the routine is either equal to the routine's name, plus the extension ".SRC", or it is equally derived from the name given in parentheses. The main chapter in this documentation where references to a particular routine occur is specified, too. The following abbreviations were used:

A ... Assembly language module.
B ... (Fortran) BLOCKDATA program.
D ... Data module.
F ... Fortran module.
R ... Subroutine or Fortran FUNCTION.
T ... Task or main routine of a task.

| | | |
|---|---|---|
| ANACNT | T-F | Analog Data Controller Task (4.5.3.1) |
| ANAINI | R-A | Analog Data Input Initialization routine (ANAINP) (4.5.2) |
| ANAINP | R-A | Analog Data Input routine (4.5.2.2) |
| ANAOPT | R-A | Analog Data Output routine (4.5.2.4) |
| ANOMAL | R-F | Anomaly Compensation routine (4.5.2.2) |
| BEEP | R-F | Beeping Routine (AUXCOM) (4.4.2) |
| BITCNT | R-A | Bit Counting routine (4.4.3.17) |
| BLKDTA | B-F | CZOOVD Data Initialization BLOCKDATA program (4.4.3) |
| CALCUL | R-F | Calculator Utility routine (4.4.3.11) |
| CHKAN1 | R-F | Operator Answer Checking routine (MENOUT) (4.4.3.4) |
| CHKANS | R-F | Operator Answer Checking routine (AUXCOM) (4.4.2) |
| CHKDTB | R-F | Check Diameter Table routine (DIACNT) (4.5.2.3) |
| CHKFNM | R-A | File Name Checking routine (4.4.3, 4.4.4.1) |
| CLEARO | R-F | Conditional Command Clearing overlay routine (4.4.3.21) |
| CLIPRL | R-F | Input Line Clearing Routine (AUXCOM) (4.4.2) |
| CLRBUF | R-F | Buffer Clearing routine |
| CLRSCR | R-F | Scrolled Screen Area Clearing routine (4.4.3.4) |
| CLSFIL | R-F | File Closing Routine (AUXCOM) (4.4.3) |
| CMFINP | T-F | Command File Input Task (4.4.6) |
| CMFOUT | T-F | Command File Output Task (4.4.7) |
| CMMDEX | T-F | Command Executor Task (4.4.4) |
| CNTRL | R-A | Control Mode Determining routine (4.4.4) |
| COMINT | T-F | Command Interpreter (4.4.3) |
| COMMEN | R-F | Comment Entry routine (4.4.3.3) |
| CONDIT | R-F | Conditional Command Entry routine (4.4.3.14) |
| CREATE | R-A | CGCS System Creation routine (CZINIT) (4.4.3) |
| CZINIT | R-A | CGCS Initialization Routine (4.4.3) |
| CZOV01 | B-F | Overlay Identification BLOCKDATA module (SETPAR) |

| | | |
|---|---|---|
| CZOV02 | B-F | Overlay Identification BLOCKDATA module (SETVAR) |
| CZOV03 | B-F | Overlay Identification BLOCKDATA module (COMMEN) |
| CZOV04 | B-F | Overlay Identification BLOCKDATA module (MENOUT) |
| CZOV05 | B-F | Overlay Identification BLOCKDATA module (OPMODE) |
| CZOV06 | B-F | Overlay Identification BLOCKDATA module (DEBUG0) |
| CZOV07 | B-F | Overlay Identification BLOCKDATA module (DEBUG1) |
| CZOV08 | B-F | Overlay Identification BLOCKDATA module (FRAME) |
| CZOV09 | B-F | Overlay Identification BLOCKDATA module (FILES) |
| CZOV10 | B-F | Overlay Identification BLOCKDATA module (REQCMF) |
| CZOV11 | B-F | Overlay Identification BLOCKDATA module (CALCUL) |
| CZOV12 | B-F | Overlay Identification BLOCKDATA module (DATAFI) |
| CZOV13 | B-F | Overlay Identification BLOCKDATA module (EXICZO) |
| CZOV14 | B-F | Overlay Identification BLOCKDATA module (CONDIT) |
| CZOV15 | B-F | Overlay Identification BLOCKDATA module (DISPLY) |
| CZOV16 | B-F | Overlay Identification BLOCKDATA module (DOCUMT) |
| CZOV17 | B-F | Overlay Identification BLOCKDATA module (DIRECT) |
| CZOV18 | B-F | Overlay Identification BLOCKDATA module (RESOVL) |
| CZOV19 | B-F | Overlay Identification BLOCKDATA module (INIDAT) |
| CZOV20 | B-F | Overlay Identification BLOCKDATA module (PLOTOV) |
| CZOV21 | B-F | Overlay Identification BLOCKDATA module (CLEARO) |
| CZOVER | D-A | System Version code |
| DASHES | R-F | Half Line Of Dashes Generating routine (FRAME) (4.4.3.8) |
| DASHLN | R-F | Full Line Of Dashes Generating routine (FRAME) (4.4.3.8) |
| DATAFI | R-F | Data File Maintenance routine (4.4.3.12) |
| DATIN | R-A | Special Input Interface routine (DATOUT) (4.3.1) |
| DATOUT | R-A | Special Output Interface routine (4.3.1) |
| DEBUG0 | R-F | DEBUG routines, part 1 (4.4.3.6) |
| DEBUG1 | R-F | DEBUG routines, part 2 (4.4.3.7) |
| DIACNT | T-F | Diameter Controller Task (4.5.2.1) |
| DIRECT | R-F | Disk Directory Display routine (4.4.3.17) |
| DISINT | R-A | Interrupt Disabling Routine (AUXASM) (4.4.4.7) |
| DISPLY | R-F | Variable Display routine (4.4.3.15) |
| DOCUMT | R-F | Documentation File Maintenance routine (4.4.3.16) |
| DSKOUT | T-F | Data File Output Task (DSKDAT) (4.4.8) |
| DUMP | R-F | Data Dump Triggering routine (DUMPDT) (4.4.4.6) |
| DUMPDT | R-F | Data Dump Generation routine (4.4.4.6) |
| ENINT | R-A | Interrupt Enabling Routine (AUXASM) (4.4.4.7) |
| ERRMSG | R-A | Error Message Output routine (AUXASM) (4.4.2) |
| EXICZO | R-F | CGCS Exit routine (4.4.3.13) |
| FILES | R-F | Output File Status Display routine (4.4.3.9) |
| FINDAD | R-A | Variable Address Finding routine (4.4.3.2) |
| FIRSTM | R-A | First module in Code area - used by MEMCHK (4.4.4.8) |
| FRAME | R-F | Console Output Mask Generation routine (4.4.3.8) |
| FRPIDC | R-A | Generic PID Controller routine (4.5.1) |
| FRSETT | R-A | Reset Timer routine (FXTIME) (4.3.2, 4.4.3) |
| FXTIME | T-A | Timer Task (4.3.2, 4.4.3) |
| FXUSIN | R-F | Initialization routine (INIT) (4.4.3) |

| | | |
|---|---|---|
| INIDAT | R-F | Initial Data Input routine (4.4.3.16) |
| INIDTA | B-F | Built-In Data Initialization BLOCKDATA program (4.4.3) |
| INIPRT | R-A | Special Output Interface routine (DATOUT) (4.3.1) |
| LOVLAY | R-F | Overlay Loading routine (AUXCOM) (4.4.3) |
| LOWPAS | R-A | Low-Pass Filtering routine (4.5.2.5) |
| LSTRAM | D-A | Dummy routine:  Last program code module |
| MAKEFN | R-A | File Name Building routine (AUXASM) (4.4.3) |
| MEASDO | T-F | Measured Data Output Task (4.4.5) |
| MEMCHK | R-A | Code Memory Checking routine (LSTRAM) (4.4.4.8) |
| MENOUT | R-F | Help Menu Output routine (4.4.3.4) |
| MESSGE | R-A | Message Output routine (AUXASM) (4.4.2) |
| MOTDIR | R-A | Motor Direction Output routine (4.5.2.3) |
| OPMODE | R-F | Operation Mode Entry routine (4.4.3.5) |
| OPNFIL | R-F | File Opening Routine (AUXCOM) (4.4.3) |
| PEEKDW | R-A | Data Retrieval routine (AUXASM) (4.4.3.1) |
| PLOTOV | R-F | Data Plotting Setup Overlay (4.4.3.20) |
| PLOTPR | R-F | Plot Data Collecting Routine (4.4.4.7) |
| PRETTA | R-A | Auxiliary Command Interpreter routine (AUXASM) (4.4.2) |
| PROMPT | R-A | Command Prompt Generation routine (AUXASM) (4.4.2) |
| QUITCM | R-F | Macro Command Quitting routine (4.4.3) |
| REACTV | R-A | Diameter Evaluation Reactivating routine (SHAPE) (4.5.2.5) |
| REQCMF | R-F | Command Output File Maintenance routine (4.4.3.10) |
| RESET | R-A | Diameter Controller Resetting routine (SHAPE) (4.5.2.4) |
| RESOVL | R-F | RESET Command Processing routine (4.4.3.18) |
| SETPAR | R-F | Parameter Setpoint Entry routine (4.4.3.1) |
| SETVAR | R-F | Variable Setpoint Entry routine (4.4.3.2) |
| SHAPE | R-A | Diameter Controller routine (4.5.2.3) |
| SHIFTB | R-F | Buffer Left Shifting routine (DEBUGO) (4.4.3.6) |
| SHIFTB | R-F | Buffer Left Shifting routine (PLOTOV) (4.4.3.20) |
| SPLITM | R-A | Mode Code Splitting routine (AUXASM) (4.4.4.1) |
| STARTP | R-A | Special Output Interface routine (DATOUT) (4.3.1) |
| STODAT | R-A | Data Storage routine (AUXASM) (4.4.4.1) |
| STRIN | R-A | Special Input Interface routine (DATOUT) (4.3.1) |
| STRIPN | R-F | Strip Binary Zeros routine (DIRECT) (4.4.3.17) |
| STROUT | R-A | Special Output Interface routine (DATOUT) (4.3.1) |
| TESTHD | R-A | Hardware Testing routine (CZINIT) (4.4.3) |
| TIMLIN | R-F | Top Of Screen Line Output routine (4.4.3, 4.4.3.8) |
| TRVMOD | D-A | Trivial Module; needed for system configuration |
| VARNM1 | B-F | Auxiliary DEBUG COMMON Block Initialization (DEBUGO) (4.4.3.6) |
| WTOUTP | R-F | MEASDO Delaying routine (MEASDO) (4.4.5) |
| XCHDSK | R-F | Disk Exchange routine (AUXCOM) (4.4.3) |

## APPENDIX E:  COMMON BLOCKS

The following table shows the Fortran COMMON blocks used in the CGCS, arranged in increasing address order. For each block, its size and the names of the routines referencing it are specified.

LOCATED IN THE MAIN COMMON AREA:

| | | |
|---|---|---|
| /ANAOUT/ | (32) | CMMDEX, PLOTPR, MEASDO, DSKOUT, ANACNT, INIDTA |
| /ANIPAR/ | (52) | ANACNT, INIDTA, BLKDTA |
| /ANOMLY/ | (8) | ANOMAL, BLKDTA |
| /ANOPAR/ | (17) | ANACNT, INIDTA, BLKDTA |
| /AUXILD/ | (62) | PLOTPR, INIDTA, BLKDTA |
| /CNDCNT/ | (1) | CMMDEX, MEASDO, INIDTA |
| /COMMEX/ | (10) | FXUSIN, COMINT, SETPAR, SETVAR, OPMODE, DEBUG0, DEBUG1, EXICZO, CONDIT, RESOVL, PLOTOV, CLEARO, CMMDEX, CMFINP, DIACNT |
| /COMMFL/ | (10) | FXUSIN, COMINT, SETPAR, SETVAR, OPMODE, DEBUG0, DEBUG1, EXICZO, CONDIT, RESOVL, PLOTOV, CLEARO, CMFINP, CMFOUT |
| /CONLIM/ | (2) | MEASDO, INIDTA, BLKDTA |
| /CRUC0P/ | (12) | DIACNT, INIDTA, BLKDTA |
| /CRUC1P/ | (12) | DIACNT, INIDTA, BLKDTA |
| /DEBUG/ | (43) | FXUSIN, CMMDEX, MEASDO, DSKOUT, INIDTA |
| /DEBUGE/ | (1) | COMINT, CMMDEX, CMFINP, INIDTA |
| /DIA10P/ | (12) | DIACNT, INIDTA, BLKDTA |
| /DIA11P/ | (12) | DIACNT, INIDTA, BLKDTA |
| /DIA20P/ | (12) | DIACNT, INIDTA, BLKDTA |
| /DIA21P/ | (12) | DIACNT, INIDTA, BLKDTA |
| /DIA30P/ | (12) | DIACNT, INIDTA, BLKDTA |

| | | |
|---|---|---|
| /DIA31P/ | (12) | DIACNT, INIDTA, BLKDTA |
| /DISKFN/ | (56) | OPNFIL, TIMLIN, FILES, REQCMF, DATAFI, DOCUMT, CMMDEX, INIDTA |
| /DOUTEX/ | (10) | FXUSIN, DSKOUT |
| /ENDBGO/ | (1) | MENOUT, OPMODE, FRAME, DIRECT, MEASDO, INIDTA |
| /INTRVL/ | (2) | EXICZO, WTOUTP, INIDTA, BLKDTA |
| /MODE/ | (1) | COMMEN, OPMODE, EXICZO, RESOVL, CMMDEX, MEASDO, DSKOUT, DIACNT, ANACNT, INIDTA |
| /OVLNM1/ | (6) | LOVLAY, CZOVxx, INIDTA, BLKDTA |
| /OVRLAY/ | (1) | COMINT, CZOVxx, FILES, INIDTA |
| /PLOTAD/ | (16) | CMMDEX, PLOTPR |
| /REALDT/ | (88) | CMMDEX, DUMPDT, PLOTPR, DIACNT, INIDTA |
| /RECORD/ | (3) | COMINT, FILES, REQCMF, CMFOUT, INIDTA |
| /RESTDO/ | (3) | FXUSIN, FRAME, EXICZO, CMMDEX, MEASDO, INIDTA |
| /RMPPAR/ | (401) | EXICZO, CMMDEX, MEASDO, INIDTA |
| /SECFLG/ | (1) | CMMDEX, ANACNT |
| /SETPT0/ | (33) | FXUSIN, SETPAR, EXICZO, INIDAT, CMMDEX, PLOTPR, MEASDO, DSKOUT, DIACNT, ANACNT, INIDTA |
| /SETPT1/ | (33) | FXUSIN, INIDAT, CMMDEX, MEASDO, DSKOUT, DIACNT, INIDTA |
| /TEMP1P/ | (12) | ANACNT, INIDTA, BLKDTA |
| /TEMP2P/ | (12) | ANACNT, INIDTA, BLKDTA |
| /TEMP3P/ | (12) | ANACNT, INIDTA, BLKDTA |
| /TEST/ | (1) | FXUSIN, ANACNT |
| /WAITEX/ | (10) | FXUSIN, QUITCM, EXICZO, INIDAT, WTOUTP |
| /XTDCNT/ | (48) | ANACNT, BLKDTA |
| /XTDDAT/ | (2) | EXICZO, DOCUMT, DUMPDT, DUMP, INIDTA |

| | | |
|---|---|---|
| /XTLSHP/ | (4) | CHKDTB, BLKDTA |

TIED TO THE DATA AREA (USED BY ASSEMBLY LANGUAGE MODULES):

MODULE FXTIME:

| | | |
|---|---|---|
| /FOTIME/ | (65) | FXUSIN, COMINT, QUITCM, COMMEN, DATAFI, EXICZO, CMMDEX, DUMPDT, DUMP, CMFINP, CMFOUT, DSKOUT, DIACNT, ANACNT |

MODULE DATOUT:

| | | |
|---|---|---|
| /IOFLAG/ | (4) | FXUSIN, COMINT, CLSFIL, OPNFIL, QUITCM, COMMEN, TIMLIN, FILES, REQCMF, DATAFI, EXICZO, DOCUMT, CMMDEX, DUMPDT, CMFINP, CMFOUT, DSKOUT, INIDTA |
| /DISKLC/ | (4) | CLSFIL, OPNFIL, FILES, REQCMF, DATAFI, DOCUMT, DIRECT, INIDTA |
| /DATE/ | (8) | FXUSIN, CLIPRL, TIMLIN, DATAFI |
| /RUNID/ | (20) | FXUSIN, TIMLIN, DATAFI |

MODULE SHAPE:

| | | |
|---|---|---|
| /ANADAT/ | (65) | FXUSIN, RESOVL, CMMDEX, PLOTPR, MEASDO, DSKOUT, DIACNT, ANACNT, INIDTA |
| /DIAMET/ | (2) | CMMDEX, PLOTPR, MEASDO, DSKOUT, DIACNT, INIDTA |
| /LENGTH/ | (2) | COMMEN, RESOVL, CMMDEX, MEASDO, DSKOUT, INIDTA |
| /SCALE/ | (72) | SETPAR, INIDAT, RESOVL, CMMDEX, PLOTPR, MEASDO, DIACNT, INIDTA, BLKDTA |
| /AUXDIA/ | (26) | INIDAT, PLOTPR, DIACNT, BLKDTA |
| /ZEROWT/ | (2) | ANACNT |
| /GROWTH/ | (4) | PLOTPR |
| /DIATAB/ | (256) | CHKDTB |

LOCATED ON TOP OF THE COMMAND INTERPRETER OVERLAY AREA:

| | | |
|---|---|---|
| /DBGCOM/ | (21) | DEBUG0, VARNM1, DEBUG1 |
| /SCONDT/ | (8) | FXUSIN, BLKDTA |

LOCATED CLOSE TO THE HIGH ADDRESS END OF THE RAM AREA (IN CONTROLLER ADDRESSABLE MEMORY)

| | | |
|---|---|---|
| /DSKBUF/ | (128) | DSKOUT, INIDTA |

## APPENDIX F: VARIABLE NAMES

### F 1: MOST IMPORTANT VARIABLES

| Name | Type | Size | Meaning |
|---|---|---|---|
| | | | Raw Analog Input Data (2 Byte Int.) |
| ITEMP1 * | I2 | 1 | Heater #1 Temperature |
| ITEMP2 * | I2 | 1 | Heater #2 Temperature |
| ITEMP3 * | I2 | 1 | Heater #3 Temperature |
| ISEEDL * | I2 | 1 | Seea Lift |
| ICRUCL * | I2 | 1 | Crucible Lift |
| ISEEDR * | I2 | 1 | Seed Rotation |
| ICRUCR * | I2 | 1 | Crucible Rotation |
| IPOUT1 * | I2 | 1 | Power Output #1 |
| IPOUT2 * | I2 | 1 | Power Output #2 |
| IPOUT3 * | I2 | 1 | Power Output #3 |
| IWEIGH * | I2 | 1 | Weight |
| IDWGHT * | I2 | 1 | Diff. Weight |
| ISEEDP * | I2 | 1 | Seed Position |
| ICRUCP * | I2 | 1 | Crucible Position |
| IBASET * | I2 | 1 | Base Temperature |
| IGASPR * | I2 | 1 | Gas Pressure |
| CONTAC * | I2 | 1 | Contact Device |
| ANALOG * | I2 | 8 | Spare Analog Channels |
| | | | Measured Analog Data (2 Byte Int.) |
| MTEMP1 + | I2 | 1 | Heater #1 Temperature |
| MTEMP2 + | I2 | 1 | Heater #2 Temperature |
| MTEMP3 + | I2 | 1 | Heater #3 Temperature |
| MSEEDL + | I2 | 1 | Seed Lift |
| MCRUCL + | I2 | 1 | Crucible Lift |
| MSEEDR + | I2 | 1 | Seed Rotation |
| MCRUCR + | I2 | 1 | Crucible Rotation |
| MPOUT1 + | I2 | 1 | Power Output #1 |
| MPOUT2 + | I2 | 1 | Power Output #2 |
| MPOUT3 + | I2 | 1 | Power Output #3 |
| MWEIGH + | I2 | 1 | Weight |
| MDWGHT + | I2 | 1 | Diff. Weight |
| MSEEDP + | I2 | 1 | Seed Position |
| MCRUCP + | I2 | 1 | Crucible Position |
| MBASET + | I2 | 1 | Base Temperature |
| MGASPR + | I2 | 1 | Gas Pressure |
| MCONTC + | I2 | 1 | Contact Device |
| MANALG + | I2 | 8 | Spare Analog Channels |

Appendix F:  Variable Names

<u>Raw Analog Output Data (2Byte Int.)</u>

| | | | | |
|---|---|---|---|---|
| PWR1IN | * | I2 | 1 | Input Power (to SCR Controller) #1 |
| PWR2IN | * | I2 | 1 | Input Power (to SCR Controller) #2 |
| PWR3IN | * | I2 | 1 | Input Power (to SCR Controller) #3 |
| SEEDLO | * | I2 | 1 | Seed Lift |
| CRUCLO | * | I2 | 1 | Crucible Lift |
| SEEDRO | * | I2 | 1 | Seed Rotation |
| CRUCRO | * | I2 | 1 | Crucible Rotation |

<u>Processed Analog Data (REAL)</u>

| | | | | |
|---|---|---|---|---|
| DIAMET | * | R | 1 | Crystal Diameter |
| TEMP1 | * | R | 1 | Heater #1 Temperature |
| TEMP2 | * | R | 1 | Heater #2 Temperature |
| TEMP3 | * | R | 1 | Heater #3 Temperature |
| SEEDL | * | R | 1 | Seed Lift |
| CRUCL | * | R | 1 | Crucible Lift |
| SEEDR | * | R | 1 | Seed Rotation |
| CRUCR | * | R | 1 | Crucible Rotation |
| POWER1 | * | R | 1 | Power Output #1 |
| POWER2 | * | R | 1 | Power Output #2 |
| POWER3 | * | R | 1 | Power Output #3 |
| WEIGHT | * | R | 1 | Weight |
| DWGHT | * | R | 1 | Diff. Weight |
| SEEDP | * | R | 1 | Seed Position |
| CRUCP | * | R | 1 | Crucible Position |
| BASTMP | * | R | 1 | Base Temperature |
| GASPR | * | R | 1 | Gas Pressure |
| PWRIN1 | * | R | 1 | Power Input (to SCR Controller) #1 |
| PWRIN2 | * | R | 1 | Power Input (to SCR Controller) #2 |
| PWRIN3 | * | R | 1 | Power Input (to SCR Controller) #3 |
| LENGTH | * | R | 1 | Crystal Length Grown |
| ADJDW | * | R | 1 | Anomaly Adjusted Diff. Weight |

<u>Current Setpoints (2 Byte Int.)</u>

| | | | | |
|---|---|---|---|---|
| STDIAM | * | I2 | 1 | Diameter |
| STTMP1 | * | I2 | 1 | Heater #1 Temperature |
| STTMP2 | * | I2 | 1 | Heater #2 Temperature |
| STTMP3 | * | I2 | 1 | Heater #3 Temperature |
| SETSL | * | I2 | 1 | Seed Lift |
| SETCL | * | I2 | 1 | Crucible Lift |
| SETSR | * | I2 | 1 | Seed Rotation |
| SETCR | * | I2 | 1 | Crucible Rotation |
| STPWRL | * | I2 | 1 | Power Limit |

PID Controller Parameters:

Seed Lift Motor

| | | | |
|---|---|---|---|
| SLGAIN | I1 | 1 | Gain |
| SLCNTL | I1 | 1 | Control |
| SLPROP | I2 | 1 | Proportional Multiplier |
| SLINT | I2 | 1 | Integral Multiplier |
| SLDIFF | I2 | 1 | Differential Multiplier |
| SLLIM | I2 | 1 | Limit |

Crucible Lift Motor

| | | | |
|---|---|---|---|
| CLGAIN | I1 | 1 | Gain |
| CLCNTL | I1 | 1 | Control |
| CLPROP | I2 | 1 | Proportional Multiplier |
| CLINT | I2 | 1 | Integral Multiplier |
| CLDIFF | I2 | 1 | Differential Multiplier |
| CLLIM | I2 | 1 | Limit |

Seed Rotation Motor

| | | | |
|---|---|---|---|
| SRGAIN | I1 | 1 | Gain |
| SRCNTL | I1 | 1 | Control |
| SRPROP | I2 | 1 | Proportional Multiplier |
| SRINT | I2 | 1 | Integral Multiplier |
| SRDIFF | I2 | 1 | Differential Multiplier |
| SRLIM | I2 | 1 | Limit |

Crucible Rotation Motor

| | | | |
|---|---|---|---|
| CRGAIN | I1 | 1 | Gain |
| CRCNTL | I1 | 1 | Control |
| CRPROP | I2 | 1 | Proportional Multiplier |
| CRINT | I2 | 1 | Integral Multiplier |
| CRDIFF | I2 | 1 | Differential Multiplier |
| CRLIM | I2 | 1 | Limit |

Temperature #1
(#2 and #3 analogously)

| | | | |
|---|---|---|---|
| T1GAIN | I1 | 1 | Gain |
| T1CNTL | I1 | 1 | Control |
| T1PROP | I2 | 1 | Proportional Multiplier |
| T1INT | I2 | 1 | Integral Multiplier |
| T1DIFF | I2 | 1 | Differential Multiplier |
| T1LIM | I2 | 1 | Limit Value |

## Appendix F: Variable Names

<u>Diameter #1</u> (controls Temp. #1)
(#3 analogously)

Main Controller

| | | | |
|---|---|---|---|
| GAIN10 | I1 | 1 | Gain |
| CNTL10 | I1 | 1 | Control |
| PROP10 | I2 | 1 | Proportional Multiplier |
| INT10 | I2 | 1 | Integral Multiplier |
| DIFF10 | I2 | 1 | Differential Multiplier |
| LIM10 | I2 | 1 | Limit Value |

Auxiliary Controller

| | | | |
|---|---|---|---|
| GAIN11 | I1 | 1 | Gain |
| CNTL11 | I1 | 1 | Control |
| PROP11 | I2 | 1 | Proportional Multiplier |
| INT11 | I2 | 1 | Integral Multiplier |
| DIFF11 | I2 | 1 | Differential Multiplier |
| LIM11 | I2 | 1 | Limit Value |

<u>Diameter #2</u> (controls Temp. #2)
(#3 analogously)

Main Controller

| | | | |
|---|---|---|---|
| GAIN20 | I1 | 1 | Gain |
| CNTL20 | I1 | 1 | Control |
| PROP20 | I2 | 1 | Proportional Multiplier |
| INT20 | I2 | 1 | Integral Multiplier |
| DIFF20 | I2 | 1 | Differential Multiplier |
| LIM20 | I2 | 1 | Limit Value |

Auxiliary Controller

| | | | |
|---|---|---|---|
| GAIN21 | I1 | 1 | Gain |
| CNTL21 | I1 | 1 | Control |
| PROP21 | I2 | 1 | Proportional Multiplier |
| INT21 | I2 | 1 | Integral Multiplier |
| DIFF21 | I2 | 1 | Differential Multiplier |
| LIM21 | I2 | 1 | Limit Value |

<u>Crucible Lift</u>

Main Controller

| | | | |
|---|---|---|---|
| COGAIN | I1 | 1 | Gain |
| COCNTL | I1 | 1 | Control |
| COPROP | I2 | 1 | Proportional Multiplier |
| COINT | I2 | 1 | Integral Multiplier |

| | | | |
|---|---|---|---|
| CODIFF | I2 | 1 | Differential Multiplier |
| COLIM | I2 | 1 | Limit Value |
| | | | Auxiliary Controller |
| C1GAIN | I1 | 1 | Gain |
| C1CNTL | I1 | 1 | Control |
| C1PROP | I2 | 1 | Proportional Multiplier |
| C1INT | I2 | 1 | Integral Multiplier |
| C1DIFF | I2 | 1 | Differential Multiplier |
| C1LIM | I2 | 1 | Limit Value |
| | | | <u>Low-Pass Filter Values (0 ... 4)</u> |
| ANIPAR(4) | I1 | | Heater #1 Temperature |
| ANIPAR(6) | I1 | | Heater #2 Temperature |
| ANIPAR(8) | I1 | | Heater #3 Temperature |
| ANIPAR(10) | I1 | | Seed Lift |
| ANIPAR(12) | I1 | | Crucible Lift |
| ANIPAR(14) | I1 | | Seed Rotation |
| ANIPAR(16) | I1 | | Crucible Rotation |
| ANIPAR(18) | I1 | | Power Output #1 |
| ANIPAR(20) | I1 | | Power Output #2 |
| ANIPAR(22) | I1 | | Power Output #3 |
| ANIPAR(24) | I1 | | Weight |
| ANIPAR(26) | I1 | | Diff. Weight |
| ANIPAR(28) | I1 | | Seed Position |
| ANIPAR(30) | I1 | | Crucible Position |
| ANIPAR(32) | I1 | | Base Temperature |
| ANIPAR(34) | I1 | | Gas Pressure |
| ANIPAR(36) | I1 | | Contact Device |
| ANIPAR(38) | I1 | | Spare Channel #1 |
| ANIPAR(40) | I1 | | Spare Channel #2 |
| ANIPAR(42) | I1 | | Spare Channel #3 |
| ANIPAR(44) | I1 | | Spare Channel #4 |
| ANIPAR(46) | I1 | | Spare Channel #5 |
| ANIPAR(48) | I1 | | Spare Channel #6 |
| ANIPAR(50) | I1 | | Spare Channel #7 |
| ANIPAR(2) | I1 | | Spare Channel #8 |
| | | | <u>Other System Control Parameters</u> |
| ANOMLY | R | 2 | Anomaly Compensation Factors |
| | | | <u>Shape Controller</u> |
| ALPHA | R | 1 | Diameter Evaluation Mode Parameter |
| XTLSHP | R | 1 | Crystal Shape Smoothing Parameter |
| CDIASQ * | R | 1 | Square of Crucible Diameter |
| SDIASQ * | R | 1 | Square of Seed Diameter |

| | | | |
|---|---|---|---|
| OXWGHT * | R | 1 | Oxide Weight |
| RHOXTL * | R | 1 | Crystal Spec. Weight (scaled) |
| RHOMLT * | R | 1 | Melt Spec. Weight (scaled) |
| RHOOXI * | R | 1 | Oxide Melt Spec. Weight (scaled) |
| SCRUCP * | I2 | 1 | Setpoint for Crucible Position |
| HEIGHT * | R | 1 | Boric Oxide Melt Height in Crucible |
| GROWTH * | R | 1 | Actual Growth Rate |
| | | | <u>Chart Recorder Output</u> |
| EXTMP1 * | I2 | 1 | Expanded Temperature 1 |
| EXTMP2 * | I2 | 1 | Expanded Temperature 2 |
| EXTMP3 * | I2 | 1 | Expanded Temperature 3 |
| EXTMPB * | I2 | 1 | Expanded Base Temperature |
| OFFST1 | R | 1 | Offset for Temperature 1 Expansion |
| OFFST2 | R | 1 | Offset for Temperature 2 Expansion |
| OFFST3 | R | 1 | Offset for Temperature 3 Expansion |
| OFFSTB | R | 1 | Offset for Base Temperature Exp. |
| RANGT1 | R | 1 | Range for Temperature 1 Expansion |
| RANGT2 | R | 1 | Range for Temperature 2 Expansion |
| RANGT3 | R | 1 | Range for Temperature 3 Expansion |
| RANGTB | R | 1 | Range for Base Temperature Exp. |
| GRRATE * | I2 | 1 | Expanded Growth Rate |
| DIAERR * | I2 | 1 | Expanded Diameter Error |
| CRPERR * | I2 | 1 | Expanded Crucible Position Error |
| ZERO | I2 | 1 | Location Holding Zero |
| | | | <u>Miscellaneous System Parameters</u> |
| TEST | I1 | 1 | Test Mode Flag |
| INTRVL | I1 | 1 | Wait Interval for Data Display (>0) |
| DUMPIN | I1 | 1 | Interval between Data Dumps |
| DUMPFL | I1 | 1 | Data Dump Request Flag |
| DIASTA | I1 | 1 | Diameter Evaluation Routine Status |
| CONLIM | I1 | 1 | Limit Value for Contact Device |
| TIME * | I2 | 1 | System Time (Seconds Counter) |
| RAMPNG * | I1 | 1 | Number of Parameters Ramped |
| CNDCNT * | I1 | 1 | Number of Conditional Commands |
| DUMMY | I2 | 8 | Scratchpad Locations |

* Read-only parameter, <u>do not change!</u>
+ Parameters can only be changed in Test mode.

**F 2: COMPLETE LIST OF VARIABLES, SORTED BY ADDRESS**

| | | | |
|---|---|---|---|
| ZERO | I2 | | LOCATION HOLDING ZERO |
| DISKIO | T | | TD FOR TASK DISKIO |
| RQTHDI | T | | TD FOR TASK RQTHDI |
| RQTHDO | T | | TD FOR TASK RQTHDO |
| RQLOAD | T | | TD FOR TASK RQLOAD |
| COMINT | T | | TD FOR TASK COMINT = RXIROM |
| RXIROM | T | | |
| PWR1IN | I2 | | INTEGER: POWER INPUT (TO SCR MODULE) |
| PWR2IN | I2 | | |
| PWR3IN | I2 | | |
| SEEDLO | I2 | | INTEGER: SEED LIFT OUTPUT |
| CRUCLO | I2 | | INTEGER: CRUCIBLE LIFT OUTPUT |
| SEEDRO | I2 | | INTEGER: SEED ROTATION OUTPUT |
| CRUCRO | I2 | | INTEGER: CRUCIBLE ROTATION OUTPUT |
| PLOTDT | I2 | 8 | INTEGER: CHART RECORDER OUTPUT DATA |
| ANAOUT | I2 | 16 | ARRAY OF INTEGER OUTPUT DATA |
| ANIPAR | I1 | 52 | PARAMETER ARRAY FOR ANALOG INPUT ROUTINE |
| ANOMLY | R | 2 | ANOMALY CORRECTION PARAMETERS (TWO REAL) |
| ANOPAR | I1 | 17 | PARAMETER ARRAY FOR ANALOG OUTPUT ROUTINE |
| OFFST1 | R | | TEMPERATURE OFFSET - HEATER TEMPERATURE I |
| OFFST2 | R | | HEATER TEMPERATURE II |
| OFFST3 | R | | HEATER TEMPERATURE III |
| OFFSTB | R | | BASE TEMPERATURE |
| RANGT1 | R | | TEMPERATURE CHART RECORDER OUTPUT RANGE - T I |
| RANGT2 | R | | HEATER TEMPERATURE II |
| RANGT3 | R | | HEATER TEMPERATURE III |
| RANGTB | R | | BASE TEMPERATURE |
| EXTMP1 | I2 | | EXPANDED HEATER TEMPERATURE I |
| EXTMP2 | I2 | | EXPANDED HEATER TEMPERATURE II |
| EXTMP3 | I2 | | EXPANDED HEATER TEMPERATURE III |
| EXTMPB | I2 | | EXPANDED BASE TEMPERATURE |
| DIAERR | I2 | | EXPANDED DIAMETER ERROR |
| CRPERR | I2 | | EXPANDED CRUCIBLE POSITION ERROR |
| GRRATE | I2 | | EXPANDED GROWTH RATE |
| DUMMY | I2 | 8 | EIGHT DUMMY LOCATIONS |
| CNDCNT | I1 | | COUNTER FOR CONDITIONAL COMMANDS |
| CONLIM | I2 | | INTEGER: LIMIT VALUE FOR CONTACT DEVICE |
| C0GAIN | I1 | | CRUCIBLE LIFT CONTROLLER ARRAY: GAIN |
| C0CNTL | I1 | | CONTROL BYTE |
| C0PROP | I2 | | PROP. MULTIPL. |
| C0INT | I2 | | INT. MULTIP. |
| C0DIFF | I2 | | DIFF. MULTIP. |
| C0LIM | I2 | | LIMIT |
| CRUC0P | I2 | 6 | CRUCIBLE LIFT CONTROLLER ARRAY |
| C1GAIN | I1 | | AUXILIARY CRUC. LIFT CONTROLLER: GAIN |
| C1CNTL | I1 | | CONTROL BYTE |
| C1PROP | I2 | | PROP. MULTIPL. |

| | | | |
|---|---|---|---|
| C1INT | I2 | | INT. MULTIPL. |
| C1DIFF | I2 | | DIFF. MULTIPL. |
| C1LIM | I2 | | LIMIT |
| CRUC1P | I2 | 6 | AUXILIARY CRUCIBLE LIFT CONTROLLER |
| GAIN10 | I1 | | MAIN DIAM. CNTL. I: GAIN |
| CNTL10 | I1 | | CONTROL BYTE |
| PROP10 | I2 | | PROP. MULTIPL. |
| INT10 | I2 | | INT. MULTIPL. |
| DIFF10 | I2 | | DIFF. MULTIPL. |
| LIM10 | I2 | | LIMIT |
| DIA10P | I2 | 6 | MAIN DIAMETER CONTROLLER I |
| GAIN11 | I1 | | AUX. DIAM. CNTL. I: GAIN |
| CNTL11 | I1 | | CONTROL BYTE |
| PROP11 | I2 | | PROP. MULTIPL. |
| INT11 | I2 | | INT. MULTIPL. |
| DIFF11 | I2 | | DIFF. MULTIPL. |
| LIM11 | I2 | | LIMIT |
| DIA11P | I2 | 6 | AUXILIARY DIAMETER CONTROLLER I |
| GAIN20 | I1 | | MAIN DIAM. CNTL. II: GAIN |
| CNTL20 | I1 | | CONTROL BYTE |
| PROP20 | I2 | | PROP. MULTIPL. |
| INT20 | I2 | | INT. MULTIPL. |
| DIFF20 | I2 | | DIFF. MULTIPL. |
| LIM20 | I2 | | LIMIT |
| DIA20P | I2 | 6 | MAIN DIAMETER CONTROLLER II |
| GAIN21 | I1 | | AUX. DIAM. CNTL. II: GAIN |
| CNTL21 | I1 | | CONTROL BYTE |
| PROP21 | I2 | | PROP. MULTIPL. |
| INT21 | I2 | | INT. MULTIPL. |
| DIFF21 | I2 | | DIFF. MULTIPL. |
| LIM21 | I2 | | LIMIT |
| DIA21P | I2 | 6 | AUXILIARY DIAMETER CONTROLLER II |
| GAIN30 | I1 | | MAIN DIAM. CNTL. III: GAIN |
| CNTL30 | I1 | | CONTROL BYTE |
| PROP30 | I2 | | PROP. MULTIPL. |
| INT30 | I2 | | INT. MULTIPL. |
| DIFF30 | I2 | | DIFF. MULTIPL. |
| LIM30 | I2 | | LIMIT |
| DIA30P | I2 | 6 | MAIN DIAMETER CONTROLLER III |
| GAIN31 | I1 | | AUX. DIAM. CNTL. III: GAIN |
| CNTL31 | I1 | | CONTROL BYTE |
| PROP31 | I2 | | PROP. MULTIPL. |
| INT31 | I2 | | INT. MULTIPL. |
| DIFF31 | I2 | | DIFF. MULTIPL. |
| LIM31 | I2 | | LIMIT |
| DIA31P | I2 | 6 | AUXILIARY DIAMETER CONTROLLER III |
| INTRVL | I2 | | INTERVAL FOR MEASURED DATA OUTPUT |
| PLOTAD | I2 | 8 | ADDRESSES OF VARIABLES SUBMITTED TO PLOT OUTPUT |
| DIAMET | R | | MEASURED DATA (REAL): DIAMETER |
| TEMP1 | R | | TEMPERATURE |

| | | | |
|---|---|---|---|
| TEMP2 | R | | |
| TEMP3 | R | | |
| SEEDL | R | | SEED LIFT |
| CRUCL | R | | CRUCIBLE LIFT |
| SEEDR | R | | SEED ROTATION |
| CRUCR | R | | CRUCIBLE ROTATION |
| POWER1 | R | | OUTPUT POWER (FROM SCR) |
| POWER2 | R | | |
| POWER3 | R | | |
| WEIGHT | R | | WEIGHT |
| DWGHT | R | | DIFF. WEIGHT |
| SEEDP | R | | SEED POSITION |
| CRUCP | R | | CRUCIBLE POSITION |
| BASTMP | R | | BASE TEMPERATURE |
| GASPR | R | | GAS PRESSURE |
| PWRIN1 | R | | POWER INPUT (TO SCR) |
| PWRIN2 | R | | |
| PWRIN3 | R | | |
| LENGTH | R | | LENGTH GROWN |
| ADJDW | R | | ADJUSTED DIFF. WEIGHT |
| REALDT | R | 22 | MEASURED DATA ARRAY (REAL) |
| RAMPNG | I1 | | NUMBER OF VARIABLES RAMPING |
| STDIAM | I2 | | CURRENT SETPOINT: DIAMETER |
| STTMP1 | I2 | | TEMPERATURE |
| STTMP2 | I2 | | |
| STTMP3 | I2 | | |
| SETSL | I2 | | SEED LIFT |
| SETCL | I2 | | CRUCIBLE LIFT |
| SETSR | I2 | | SEED ROTATION |
| SETCR | I2 | | CRUCIBLE ROTATION |
| STPWRL | I2 | | POWER LIMIT |
| SETPTO | I2 | 9 | CURRENT SETPOINT ARRAY (INTEGER) |
| T1GAIN | I1 | | TEMP. CNTL. I: GAIN |
| T1CNTL | I1 | | CONTROL BYTE |
| T1PROP | I2 | | PROP. MULTIPL. |
| T1INT | I2 | | INT. MULTIPL. |
| T1DIFF | I2 | | DIFF. MULTIPL. |
| T1LIM | I2 | | LIMIT |
| TEMP1P | I2 | 6 | TEMPERATURE CONTROLLER I |
| T2GAIN | I1 | | TEMP. CNTL. II: GAIN |
| T2CNTL | I1 | | CONTROL BYTE |
| T2PROP | I2 | | PROP. MULTIPL. |
| T2INT | I2 | | INT. MULTIPL. |
| T2DIFF | I2 | | DIFF. MULTIPL. |
| T2LIM | I2 | | LIMIT |
| TEMP2P | I2 | 6 | TEMPERATURE CONTROLLER II |
| T3GAIN | I1 | | TEMP. CNTL. III: GAIN |
| T3CNTL | I1 | | CONTROL BYTE |
| T3PROP | I2 | | PROP. MULTIPL. |
| T3INT | I2 | | INT. MULTIPL. |

| | | | |
|---|---|---|---|
| T3DIFF | I2 | | DIFF. MULTIPL. |
| T2LIM | I2 | | LIMIT |
| TEMP3P | I2 | 6 | TEMPERATURE CONTROLLER III |
| TEST | I1 | | TEST MODE FLAG |
| SLGAIN | I1 | | SEED LIFT CNTL.: GAIN |
| SLCNTL | I1 | | CONTROL BYTE |
| SLPROP | I2 | | PROP. MULTIPL. |
| SLINT | I2 | | INT. MULTIPL. |
| SLDIFF | I2 | | DIFF. MULTIPL. |
| SLLIM | I2 | | LIMIT |
| SEEDLP | I2 | 6 | SEED LIFT CONTROLLER |
| CLGAIN | I1 | | CRUC. LIFT CNTL.: GAIN |
| CLCNTL | I1 | | CONTROL BYTE |
| CLPROP | I2 | | PROP. MULTIPL. |
| CLINT | I2 | | INT. MULTIPL. |
| CLDIFF | I2 | | DIFF. MULTIPL. |
| CLLIM | I2 | | LIMIT |
| CRUCLP | I2 | 6 | CRUC. LIFT CONTROLLER |
| SRGAIN | I1 | | SEED ROT. CNTL.: GAIN |
| SRCNTL | I1 | | CONTROL BYTE |
| SRPROP | I2 | | PROP. MULTIPL. |
| SRINT | I2 | | INT. MULTIPL. |
| SRDIFF | I2 | | DIFF. MULTIPL. |
| SRLIM | I2 | | LIMIT |
| SEEDRP | I2 | 6 | SEED ROT CONTROLLER |
| CRGAIN | I1 | | CRUC. ROT. CNTL.: GAIN |
| CRCNTL | I1 | | CONTROL BYTE |
| CRPROP | I2 | | PROP. MULTIPL. |
| CRINT | I2 | | INT. MULTIPL. |
| CRDIFF | I2 | | DIFF. MULTIPL. |
| CRLIM | I2 | | LIMIT |
| CRUCRP | I2 | 6 | CRUC. ROT. CONTROLLER |
| DUMPIN | I1 | | INTERVAL FOR DATA DUMPS |
| DUMPFL | I1 | | DUMP FLAG |
| XTLSHP | R | | CRYSTAL SHAPE PARAMETER |
| CMMDEX | T | | TD FOR CMMDEX |
| MEASDO | T | | TD FOR MEASDO |
| CMFINP | T | | TD FOR CMFINP |
| CMFOUT | T | | TD FOR CMFOUT |
| DSKOUT | T | | TD FOR DISKOUT |
| DIACNT | T | | TD FOR DIACNT |
| ANACNT | T | | TD FOR ANACNT |
| ALARMF | I1 | | ALARM TIMER INTERRUPT FLAG |
| TIME | I2 | | SYSTEM TIME (INTEGER) |
| DIFFTM | I2 | | DIFFERENTIAL TIME FOR MACRO EXECUTION |
| DTINTV | I1 | | DATA FILE UPDATING INTERVAL |
| TIMSET | I2 | | SETPOINT FOR ALARM TIMER (MACRO EXECUTION) |
| IOFLAG | I1 | 4 | I/O FLAG ARRAY |
| ITEMP1 | I2 | | MEASURED DATA (INTEGER): TEMPERATURE |
| ITEMP2 | I2 | | |

| | | | |
|---|---|---|---|
| ITEMP3 | I2 | | |
| ISEEDL | I2 | | SEED LIFT SPEED |
| ICRUCL | I2 | | CRUCIBLE LIFT SPEED |
| ISEEDR | I2 | | SEED ROTATION |
| ICRUCR | I2 | | CRUCIBLE ROTATION |
| IPOUT1 | I2 | | POWER OUTPUT (FROM SCR) |
| IPOUT2 | I2 | | |
| IPOUT3 | I2 | | |
| IWEIGH | I2 | | WEIGHT |
| IDWGHT | I2 | | DIFF. WEIGHT |
| ISEEDP | I2 | | SEED POSITION |
| ICRUCP | I2 | | CRUCIBLE POSITION |
| IBASET | I2 | | BASE TEMPERATURE |
| IGASPR | I2 | | GAS PRESSURE |
| CONTAC | I2 | | CONTACT |
| ANALOG | I2 | 8 | EIGHT SPARE ANALOG CHANNELS (INTEGER) |
| ANADAT | I2 | 25 | COMPLETE ARRAY OF ANALOG DATA (INTEGER) |
| IDIAMT | I2 | | CRYSTAL DIAMETER (INTEGER) |
| ILENGT | I2 | | LENGTH GROWN (INTEGER) |
| SCADIA | R | | SCALING FACTORS: DIAMETER |
| SCATMP | R | 3 | TEMPERATURES |
| SCAMOT | R | 4 | MOTORS |
| SCAPWO | R | 3 | POWER OUTPUT |
| SCAWGT | R | | WEIGHT |
| SCADWT | R | | DIFFERENTIAL WEIGHT |
| SCAPOS | R | 2 | POSITION |
| SCABST | R | | BASE TEMPERATURE |
| SCAGAS | R | | GAS PRESSURE |
| SCAPWR | R | | POWER INPUT AND LIMIT |
| SCALE | R | 18 | ARRAY OF SCALING FACTORS |
| CDIASQ | R | | SQUARE OF CRUCIBLE DIAMETER |
| SDIASQ | R | | SQUARE OF SEED DIAMETER |
| OXWGHT | R | | BORIC OXIDE WEIGHT |
| RHOXTL | R | | DENSITY: CRYSTAL |
| RHOMLT | R | | MELT |
| RHOOXI | R | | OXIDE |
| SCRUCP | I2 | | SETPOINT FOR CRUC. POSITION (INTEGER) |
| ZEROWT | I2 | | WEIGHT ZEROING OFFSET |
| GROWTH | R | | ACTUAL GROWTH RATE |
| ALPHA | R | | CORRECTION FACTOR FOR GROWTH PATE |
| DIATAB | R | 64 | DIAMETER SQUARES TABLE |
| IHEIGH | I2 | | MELT HEIGHT (SCALED AS LENGTH) |
| OLDLEN | I2 | | LENGTH AT LAST SLICE BOUNDARY |
| DIFFLG | I2 | | HEIGHT OF CURRENT SLICE |
| RDWGHT | R | | (ADJUSTED) DIFFERENTIAL WEIGHT (FLOATING-POINT) |
| RHOOXA | R | | ADJUSTED OXIDE DENSITY |
| DIA1SQ | R | | SQUARE OF DIAMETER AT OXIDE SURFACE |
| DIA2SQ | R | | SQUARE OF DIAMETER AT MELT SURFACE |
| HEIGHT | R | | BORIC OXIDE HEIGHT IN CRUCIBLE (REAL) |
| RCRSET | R | | CRUCIBLE POSITION SETPOINT (REAL) |

| | | | |
|---|---|---|---|
| VOLSUM | R | | SUM OF VOLUMES IN CURRENT SLICE (UNSCALED) |
| OXIVOL | R | | VOLUME OF BORIC OXIDE MELT |
| CORRVL | R | | OXIDE VOLUME CORRECTION |
| BETA | R | | CORRECTION FACTOR |
| RDLIFT | R | | SEED - CRUCIBLE LIFT SPEEDS |
| RLNGTH | R | | UNSCALED LENGTH |
| PRLNGT | R | | UNSCALED LENGTH DURING PREVIOUS PASS |
| INICRP | R | | CRUCIBLE POSITION AT RESET |
| ADJLEN | R | | LENGTH ADJUSTMENT PARAMETER |
| DIASTA | I1 | | DIAMETER CONTROLLER STATUS |
| LOOPCT | I1 | | LOOP COUNTER LOCATION |
| OXOVFL | I1 | | OXIDE HEIGHT OVERFLOW FLAG |
| MTEMP1 | I2 | | PRIMARY MEASURED DATA (INTEGER): TEMPERATURE |
| MTEMP2 | I2 | | |
| MTEMP3 | I2 | | |
| MSEEDL | I2 | | SEED LIFT SPEED |
| MCRUCL | I2 | | CRUCIBLE LIFT SPEED |
| MSEEDR | I2 | | SEED ROTATION |
| MCRUCR | I2 | | CRUCIBLE ROTATION |
| MPOUT1 | I2 | | POWER OUTPUT (FROM SCR) |
| MPOUT2 | I2 | | |
| MPOUT3 | I2 | | |
| MWEIGH | I2 | | WEIGHT |
| MDWGHT | I2 | | DIFF. WEIGHT |
| MSEEDP | I2 | | SEED POSITION |
| MCRUCP | I2 | | CRUCIBLE POSITION |
| MBASET | I2 | | BASE TEMPERATURE |
| MGASPR | I2 | | GAS PRESSURE |
| MCONTC | I2 | | CONTACT |
| MANALG | I2 | 25 | ANALOG DATA INPUT ARRAY |

## F 3: VARIABLE ADDRESSES FOR CGCS VERSIONS 1.6 THROUGH 2.3

| V1.6 | V2.0 | V2.1 | V2.2 | V2.3 | | | |
|---|---|---|---|---|---|---|---|
| ---- | 1FF6 | 1FF6 | 1FF6 | 1FF6 | ZERO | I2 | LOCATION HOLDING ZERO |
| 2136 | 2136 | 2136 | 2136 | 2136 | DISKIO | T | TD FOR TASK DISKIO |
| 214A | 214A | 214A | 214A | 214A | RQTHDI | T | TD FOR TASK RQTHDI |
| 215E | 215E | 215E | 215E | 215E | RQTHDO | T | TD FOR TASK RQTHDO |
| 2172 | 2172 | 2172 | 2172 | 2172 | RQLOAD | T | TD FOR TASK RQLOAD |
| 2186 | 2186 | 2186 | 2186 | 2186 | COMINT | T | TD FOR TASK COMINT = RXIROM |
| 2186 | 2186 | 2186 | 2186 | 2186 | RXIROM | T | |
| 2800 | 2800 | 2800 | 2800 | 2800 | PWR1IN | I2 | INTEGER: POWER INPUT (TO SCR MODULE) |
| 2802 | 2802 | 2802 | 2802 | 2802 | PWR2IN | I2 | |
| 2804 | 2804 | 2804 | 2804 | 2804 | PWR3IN | I2 | |

| V1.6 | V2.0 | V2.1 | V2.2 | V2.3 | | | | |
|---|---|---|---|---|---|---|---|---|
| 2806 | 2806 | 2806 | 2806 | 2806 | SEEDLO | I2 | | INTEGER: SEED LIFT OUTPUT |
| 2808 | 2808 | 2808 | 2808 | 2808 | CRUCLO | I2 | | INTEGER: CRUCIBLE LIFT OUTPUT |
| 280A | 280A | 280A | 280A | 280A | SEEDRO | I2 | | INTEGER: SEED ROTATION OUTPUT |
| 280C | 280C | 280C | 280C | 280C | CRUCRO | I2 | | INTEGER: CRUCIBLE ROTATION OUTPUT |
| ---- | 280E | 280E | 280E | 280E | PLOTDT | I2 | 8 | INTEGER: OUTPUT TO CHART RECORDER |
| 2800 | 2800 | 2800 | 2800 | 2800 | ANAOUT | I2 | 16 | ARRAY OF INTEGER OUTPUT DATA |
| 280E | 2820 | 2820 | 2820 | 2820 | ANIPAR | I1 | 52 | PARAMETER ARRAY FOR ANALOG INPUT |
| 2842 | 2854 | 2854 | 2854 | 2854 | ANOMLY | R | 2 | ANOMALY CORRECTION PARAMETERS (REAL) |
| 284A | 285C | 285C | 285C | 285C | ANOPAR | I1 | 17 | PARAMETER ARRAY FOR ANALOG OUTPUT |
| ---- | 286D | 286D | 286D | 286D | OFFST1 | R | | TEMPERATURE OFFSET - HEATER TEMP I |
| ---- | 2871 | 2871 | 2871 | 2871 | OFFST2 | R | | HEATER TEMPERATURE II |
| ---- | 2875 | 2875 | 2875 | 2875 | OFFST3 | R | | HEATER TEMPERATURE III |
| ---- | 2879 | 2879 | 2879 | 2879 | OFFSTB | R | | BASE TEMPERATURE |
| ---- | ---- | 287D | 287D | 287D | RANGT1 | R | | TEMP. CHART RECORDER OUTPUT RANGE - I |
| ---- | ---- | 2881 | 2881 | 2881 | RANGT2 | R | | HEATER TEMPERATURE II |
| ---- | ---- | 2885 | 2885 | 2885 | RANGT3 | R | | HEATER TEMPERATURE III |
| ---- | ---- | 2889 | 2889 | 2889 | RANGTB | R | | BASE TEMPERATURE |
| ---- | 287D | 288D | 288D | 288D | EXTMP1 | I2 | | EXPANDED HEATER I TEMPERATURE |
| ---- | 287F | 288F | 288F | 288F | EXTMP2 | I2 | | HEATER II |
| ---- | 2881 | 2891 | 2891 | 2891 | EXTMP3 | I2 | | HEATER III |
| ---- | 2883 | 2893 | 2893 | 2893 | EXTMPB | I2 | | BASE |
| ---- | 2885 | 2895 | 2895 | 2895 | DIAERR | I2 | | DIAMETER ERROR (FOR PLOT) |
| ---- | 2887 | 2897 | 2897 | 2897 | CRPERR | I2 | | CRUCIBLE POSITION ERROR (FOR PLOT) |
| ---- | 2889 | 2899 | 2899 | 2899 | GRRATE | I2 | | GROWTH RATE (FOR PLOT) |
| ---- | 288B | 289B | 289B | 289B | DUMMY | I2 | 8 | EIGHT DUMMY LOCATIONS |
| 2852 | 289B | 28AB | 28AB | 28AB | CNDCNT | I1 | | COUNTER FOR CONDITIONAL COMMANDS |
| 2867 | 28B0 | 28C0 | 28C0 | 28C0 | CONLIM | I2 | | INTEGER: LIMIT VALUE FOR CONTACT DEV. |
| 2869 | 28B2 | 28C2 | 28C2 | 28C2 | COGAIN | I1 | | CRUCIBLE LIFT CONTROLLER ARRAY: GAIN |
| 286A | 28B3 | 28C3 | 28C3 | 28C3 | COCNTL | I1 | | CONTROL BYTE |
| 286D | 28B6 | 28C6 | 28C6 | 28C6 | COPROP | I2 | | PROP. MULTIPL. |
| 286F | 28B8 | 28C8 | 28C8 | 28C8 | COINT | I2 | | INT. MULTIP. |
| 2871 | 28BA | 28CA | 28CA | 28CA | CODIFF | I2 | | DIFF. MULTIP. |
| 2873 | 28BC | 28CC | 28CC | 28CC | COLIM | I2 | | LIMIT |
| 2869 | 28B2 | 28C2 | 28C2 | 28C2 | CRUCOP | I2 | 6 | CRUCIBLE LIFT CONTROLLER ARRAY |
| 2875 | 28BE | 28CE | 28CE | 28CE | C1GAIN | I1 | | AUXILIARY CRUC. LIFT CONTROLLER: GAIN |
| 2876 | 28BF | 28CF | 28CF | 28CF | C1CNTL | I1 | | CONTROL BYTE |
| 2879 | 28C2 | 28D2 | 28D2 | 28D2 | C1PROP | I2 | | PROP. MULTIPL. |
| 287B | 28C4 | 28D4 | 28D4 | 28D4 | C1INT | I2 | | INT. MULTIPL. |
| 287D | 28C6 | 28D6 | 28D6 | 28D6 | C1DIFF | I2 | | DIFF. MULTIPL. |
| 287F | 28C8 | 28D8 | 28D8 | 28D8 | C1LIM | I2 | | LIMIT |
| 2875 | 28BE | 28CE | 28CE | 28CE | CRUC1P | I2 | 6 | AUXILIARY CRUCIBLE LIFT CONTROLLER |
| 28AD | 28F6 | 2906 | 2906 | 2906 | GAIN10 | I1 | | MAIN DIAM. CNTL. I GAIN |
| 28AE | 28F7 | 2907 | 2907 | 2907 | CNTL10 | I1 | | CONTROL BYTE |
| 28B1 | 28FA | 290A | 290A | 290A | PROP10 | I2 | | PROP. MULTIPL. |
| 28B3 | 28FC | 290C | 290C | 290C | INT10 | I2 | | INT. MULTIPL. |
| 28B5 | 28FE | 290E | 290E | 290E | DIFF10 | I2 | | DIFF. MULTIPL. |
| 28B7 | 2900 | 2910 | 2910 | 2910 | LIM10 | I2 | | LIMIT |
| 28AD | 28F6 | 2906 | 2906 | 2906 | DIA1DP | I2 | 6 | MAIN DIAMETER CONTROLLER I |
| 28B9 | 2902 | 2912 | 2912 | 2912 | GAIN11 | I1 | | AUX. DIAM. CNTL. I GAIN |

| V1.6 | V2.0 | V2.1 | V2.2 | V2.3 | | | | |
|---|---|---|---|---|---|---|---|---|
| 28BA | 2903 | 2913 | 2913 | 2913 | CNTL11 | I1 | | CONTROL BYTE |
| 28BD | 2906 | 2916 | 2916 | 2916 | PROP11 | I2 | | PROP. MULTIPL. |
| 28BF | 2908 | 2918 | 2918 | 2918 | INT11 | I2 | | INT. MULTIPL. |
| 28C1 | 290A | 291A | 291A | 291A | DIFF11 | I2 | | DIFF. MULTIPL. |
| 28C3 | 290C | 291C | 291C | 291C | LIM11 | I2 | | LIMIT |
| 28B9 | 2902 | 2912 | 2912 | 2912 | DIA11P | I2 | 6 | AUXILIARY DIAMETER CONTROLLER I |
| 28C5 | 290E | 291E | 291E | 291E | GAIN20 | I1 | | MAIN DIAM. CNTL. II GAIN |
| 28C6 | 290F | 291F | 291F | 291F | CNTL20 | I1 | | CONTROL BYTE |
| 28C9 | 2912 | 2922 | 2922 | 2922 | PROP20 | I2 | | PROP. MULTIPL. |
| 28CB | 2914 | 2924 | 2924 | 2924 | INT20 | I2 | | INT. MULTIPL. |
| 28CD | 2916 | 2926 | 2926 | 2926 | DIFF20 | I2 | | DIFF. MULTIPL. |
| 28CF | 2918 | 2928 | 2928 | 2928 | LIM20 | I2 | | LIMIT |
| 28C5 | 290E | 291E | 291E | 291E | DIA20P | I2 | 6 | MAIN DIAMETER CONTROLLER II |
| 28D1 | 291A | 292A | 292A | 292A | GAIN21 | I1 | | AUX. DIAM. CNTL. II GAIN |
| 28D2 | 291B | 292B | 292B | 292B | CNTL21 | I1 | | CONTROL BYTE |
| 28D5 | 291E | 292E | 292E | 292E | PROP21 | I2 | | PROP. MULTIPL. |
| 28D7 | 2920 | 2930 | 2930 | 2930 | INT21 | I2 | | INT. MULTIPL. |
| 28D9 | 2922 | 2932 | 2932 | 2932 | DIFF21 | I2 | | DIFF. MULTIPL. |
| 28DB | 2924 | 2934 | 2934 | 2934 | LIM21 | I2 | | LIMIT |
| 28D1 | 291A | 292A | 292A | 292A | DIA21P | I2 | 6 | AUXILIARY DIAMETER CONTROLLER II |
| 28DD | 2926 | 2936 | 2936 | 2936 | GAIN30 | I1 | | MAIN DIAM. CNTL. III GAIN |
| 28DE | 2927 | 2937 | 2937 | 2937 | CNTL30 | I1 | | CONTROL BYTE |
| 28E1 | 292A | 293A | 293A | 293A | PROP30 | I2 | | PROP. MULTIPL. |
| 28E3 | 292C | 293C | 293C | 293C | INT30 | I2 | | INT. MULTIPL. |
| 28E5 | 292E | 293E | 293E | 293E | DIFF30 | I2 | | DIFF. MULTIPL. |
| 28E7 | 2930 | 2940 | 2940 | 2940 | LIM30 | I2 | | LIMIT |
| 28DD | 2926 | 2936 | 2936 | 2936 | DIA30P | I2 | 6 | MAIN DIAMETER CONTROLLER III |
| 28E9 | 2932 | 2942 | 2942 | 2942 | GAIN31 | I1 | | AUX. DIAM. CNTL. III GAIN |
| 28EA | 2933 | 2943 | 2943 | 2943 | CNTL31 | I1 | | CONTROL BYTE |
| 28ED | 2936 | 2946 | 2946 | 2946 | PROP31 | I2 | | PROP. MULTIPL. |
| 28EF | 2938 | 2948 | 2948 | 2948 | INT31 | I2 | | INT. MULTIPL. |
| 28F1 | 293A | 294A | 294A | 294A | DIFF31 | I2 | | DIFF. MULTIPL. |
| 28F3 | 293C | 294C | 294C | 294C | LIM31 | I2 | | LIMIT |
| 28E9 | 2932 | 2942 | 2942 | 2942 | DIA31P | I2 | 6 | AUXILIARY DIAMETER CONTROLLER III |
| 293C | 2985 | 2991 | 2991 | 2991 | INTRVL | I2 | | INTERVAL FOR MEASUREMENT DATA OUTPUT |
| .... | 298F | 299B | 299B | 299B | PLOTAD | I2 | 8 | ADDRESSES OF CHART RECORDER OUTPUT |
| 2946 | 299F | 29AB | 29AB | 29AB | DIAMET | R | | MEASURED DATA (REAL): DIAMETER |
| 294A | 29A3 | 29AF | 29AF | 29AF | TEMP1 | R | | TEMPERATURE |
| 294E | 29A7 | 29B3 | 29B3 | 29B3 | TEMP2 | R | | |
| 2952 | 29AB | 29B7 | 29B7 | 29B7 | TEMP3 | R | | |
| 2956 | 29AF | 29BB | 29BB | 29BB | SEEDL | R | | SEED LIFT |
| 295A | 29B3 | 29BF | 29BF | 29BF | CRUCL | R | | CRUCIBLE LIFT |
| 295E | 29B7 | 29C3 | 29C3 | 29C3 | SEEDR | R | | SEED ROTATION |
| 2962 | 29BB | 29C7 | 29C7 | 29C7 | CRUCR | R | | CRUCIBLE ROTATION |
| 2966 | 29BF | 29CB | 29CB | 29CB | POWER1 | R | | OUTPUT POWER (FROM SCR) |
| 296A | 29C3 | 29CF | 29CF | 29CF | POWER2 | R | | |
| 296E | 29C7 | 29D3 | 29D3 | 29D3 | POWER3 | R | | |
| 2972 | 29CB | 29D7 | 29D7 | 29D7 | WEIGHT | R | | WEIGHT |
| 2976 | 29CF | 29DB | 29DB | 29DB | DWGHT | R | | DIFFERENTIAL WEIGHT |

| V1.6 | V2.0 | V2.1 | V2.2 | V2.3 | | | | |
|---|---|---|---|---|---|---|---|---|
| 297A | 29D3 | 29DF | 29DF | 29DF | SEEOP | R | | SEED POSITION |
| 297E | 29D7 | 29E3 | 29E3 | 29E3 | CRUCP | R | | CRUCIBLE POSITION |
| 2982 | 29DB | 29E7 | 29E7 | 29E7 | BASTMP | R | | BASE TEMPERATURE |
| 2986 | 29DF | 29EB | 29EB | 29EB | GASPR | R | | GAS PRESSURE |
| 298A | 29E3 | 29EF | 29EF | 29EF | PWRIN1 | R | | POWER INPUT (TO SCR) |
| 298E | 29E7 | 29F3 | 29F3 | 29F3 | PWRIN2 | R | | |
| 2992 | 29EB | 29F7 | 29F7 | 29F7 | PWRIN3 | R | | |
| 2996 | 29EF | 29FB | 29FB | 29FB | LENGTH | R | | LENGTH GROWN |
| 299A | 29F3 | 29FF | 29FF | 29FF | ADJDW | R | | ADJUSTED DIFF. WEIGHT |
| 2946 | 299F | 29AB | 29AB | 29AB | REALDT | R | 22 | MEASURED DATA ARRAY (REAL) |
| 29A4 | 29FD | 2A09 | 2A09 | 2A09 | RAMPNG | I1 | | NUMBER OF VARIABLES RAMPING |
| 2A55 | 2B9E | 2BAA | 2BAA | 2BAA | STDIAM | I2 | | CURRENT SETPOINT DIAMETER |
| 2A57 | 2BA0 | 2BAC | 2BAC | 2BAC | STTMP1 | I2 | | TEMPERATURE |
| 2A59 | 2BA2 | 2BAE | 2BAE | 2BAE | STTMP2 | I2 | | |
| 2A5B | 2BA4 | 2BB0 | 2BB0 | 2BB0 | STTMP3 | I2 | | |
| 2A50 | 2BA6 | 2BB2 | 2BB2 | 2BB2 | SETSL | I2 | | SEED LIFT |
| 2A5F | 2BA8 | 2BB4 | 2BB4 | 2BB4 | SETCL | I2 | | CRUCIBLE LIFT |
| 2A61 | 2BAA | 2BB6 | 2BB6 | 2BB6 | SETSR | I2 | | SEED ROTATION |
| 2A63 | 2BAC | 2BB8 | 2BB8 | 2BB8 | SETCR | I2 | | CRUCIBLE ROTATION |
| 2A65 | 2BAE | 2BBA | 2BBA | 2BBA | STPWRL | I2 | | POWER LIMIT |
| 2A55 | 2B9E | 2BAA | 2BAA | 2BAA | SETPTO | I2 | 9 | CURRENT SETPOINT ARRAY (INTEGER) |
| 2A88 | 2BD1 | 2BDD | 2BDD | 2BDD | T1GAIN | I1 | | TEMP. CNTL. I GAIN |
| 2A89 | 2BD2 | 2BDE | 2BDE | 2BDE | T1CNTL | I1 | | CONTROL BYTE |
| 2A8C | 2BD5 | 2BE1 | 2BE1 | 2BE1 | T1PROP | I2 | | PROP. MULTIPL. |
| 2A8E | 2BD7 | 2BE3 | 2BE3 | 2BE3 | T1INT | I2 | | INT. MULTIPL. |
| 2A90 | 2BD9 | 2BE5 | 2BE5 | 2BE5 | T1DIFF | I2 | | DIFF. MULTIPL. |
| 2A92 | 2BDB | 2BE7 | 2BE7 | 2BE7 | T1LIM | I2 | | LIMIT |
| 2A88 | 2B01 | 2BDD | 2BD0 | 2BDD | TEMP1P | I2 | 6 | TEMPERATURE CONTROLLER I |
| 2A94 | 2BDD | 2BE9 | 2BE9 | 2BE9 | T2GAIN | I1 | | TEMP. CNTL. II GAIN |
| 2A95 | 2BDE | 2BEA | 2BEA | 2BEA | T2CNTL | I1 | | CONTROL BYTE |
| 2A98 | 2BE1 | 2BE0 | 2BED | 2BED | T2PROP | I2 | | PROP. MULTIPL. |
| 2A9A | 2BE3 | 2BEF | 2BEF | 2BEF | T2INT | I2 | | INT. MULTIPL. |
| 2A9C | 2BE5 | 2BF1 | 2BF1 | 2BF1 | T2DIFF | I2 | | DIFF. MULTIPL. |
| 2A9E | 2BE7 | 2BF3 | 2BF3 | 2BF3 | T2LIM | I2 | | LIMIT |
| 2A94 | 2B0D | 2BE9 | 2BE9 | 2BE9 | TEMP2P | I2 | 6 | TEMPERATURE CONTROLLER II |
| 2AA0 | 2BE9 | 2BF5 | 2BF5 | 2BF5 | T3GAIN | I1 | | TEMP. CNTL. III GAIN |
| 2AA1 | 2BEA | 2BF6 | 2BF6 | 2BF6 | T3CNTL | I1 | | CONTROL BYTE |
| 2AA4 | 2BED | 2BF9 | 2BF9 | 2BF9 | T3PROP | I2 | | PROP. MULTIPL. |
| 2AA6 | 2BEF | 2BFB | 2BFB | 2BFB | T3INT | I2 | | INT. MULTIPL. |
| 2AA8 | 2BF1 | 2BFD | 2BFD | 2BFD | T3DIFF | I2 | | DIFF. MULTIPL. |
| 2AAA | 2BF3 | 2BFF | 2BFF | 2BFF | T2LIM | I2 | | LIMIT |
| 2AA0 | 2BE9 | 2BF5 | 2BF5 | 2BF5 | TEMP3P | I2 | 6 | TEMPERATURE CONTROLLER III |
| .... | 2BF5 | 2C01 | 2C01 | 2C01 | TEST | I1 | | TEST MODE FLAG |
| 2AB6 | 2C00 | 2C0C | 2C0C | 2C0C | SLGAIN | I1 | | SEED LIFT CNTL.: GAIN |
| 2AB7 | 2C01 | 2C0D | 2C0D | 2C0D | SLCNTL | I1 | | CNTL |
| 2ABA | 2C04 | 2C10 | 2C10 | 2C10 | SLPROP | I2 | | PROP. MULTIPLIER |
| 2ABC | 2C06 | 2C12 | 2C12 | 2C12 | SLINT | I2 | | INT. MULTIPL. |
| 2ABE | 2C08 | 2C14 | 2C14 | 2C14 | SLDIFF | I2 | | DIFF. MULTIPL. |
| 2AC0 | 2C0A | 2C16 | 2C16 | 2C16 | SLLIM | I2 | | LIMIT |

| V1.6 | V2.D | V2.1 | V2.2 | V2.3 | | | | |
|---|---|---|---|---|---|---|---|---|
| 2AB6 | 2CDD | 2CDC | 2CDC | 2CDC | SEEDLP | I2 | 6 | SEED LIFT CONTROLLER |
| 2AC2 | 2CDC | 2C18 | 2C18 | 2C18 | CLGAIN | I1 | | CRUC LIFT CNTL.: GAIN |
| 2AC3 | 2C0D | 2C19 | 2C19 | 2C19 | CLCNTL | I1 | | CNTL |
| 2AC6 | 2C10 | 2C1C | 2C1C | 2C1C | CLPROP | I2 | | PROP. MULTIPLIER |
| 2AC8 | 2C12 | 2C1E | 2C1E | 2C1E | CLINT | I2 | | INT. MULTIPL. |
| 2ACA | 2C14 | 2C20 | 2C20 | 2C20 | CLDIFF | I2 | | DIFF. MULTIPL. |
| 2ACC | 2C16 | 2C22 | 2C22 | 2C22 | CLLIM | I2 | | LIMIT |
| 2AC2 | 2CDC | 2C18 | 2C18 | 2C18 | CRUCLP | I2 | 6 | CRUC LIFT CONTROLLER |
| 2ACE | 2C18 | 2C24 | 2C24 | 2C24 | SRGAIN | I1 | | SEED ROT CNTL.: GAIN |
| 2ACF | 2C19 | 2C25 | 2C25 | 2C25 | SRCNTL | I1 | | CNTL |
| 2AD2 | 2C1C | 2C28 | 2C28 | 2C28 | SRPROP | I2 | | PROP. MULTIPLIER |
| 2AD4 | 2C1E | 2C2A | 2C2A | 2C2A | SRINT | I2 | | INT. MULTIPL. |
| 2AD6 | 2C20 | 2C2C | 2C2C | 2C2C | SRDIFF | I2 | | DIFF. MULTIPL. |
| 2AD8 | 2C22 | 2C2E | 2C2E | 2C2E | SRLIM | I2 | | LIMIT |
| 2ACE | 2C18 | 2C24 | 2C24 | 2C24 | SEEDRP | I2 | 6 | SEED ROT CONTROLLER |
| 2ADA | 2C24 | 2C30 | 2C30 | 2C3D | CRGAIN | I1 | | CRUC ROT CNTL.: GAIN |
| 2ADB | 2C25 | 2C31 | 2C31 | 2C31 | CRCNTL | I1 | | CNTL |
| 2ADE | 2C28 | 2C34 | 2C34 | 2C34 | CRPROP | I2 | | PROP. MULTIPLIER |
| 2AE0 | 2C2A | 2C36 | 2C36 | 2C36 | CRINT | I2 | | INT. MULTIPL. |
| 2AE2 | 2C2C | 2C38 | 2C38 | 2C38 | CRDIFF | I2 | | DIFF. MULTIPL. |
| 2AE4 | 2C2E | 2C3A | 2C3A | 2C3A | CRLIM | I2 | | LIMIT |
| 2AB6 | 2C24 | 2C30 | 2C3D | 2C30 | CRUCRP | I2 | 6 | CRUC ROT CONTROLLER |
| 2AE6 | 2C3D | 2C3C | 2C3C | 2C3C | DUMPIN | I1 | | INTERVAL FOR DATA DUMPS |
| 2AE7 | 2C31 | 2C3D | 2C3D | 2C3D | DUMPFL | I1 | | DUMP FLAG |
| .... | .... | .... | 2C3E | 2C3E | XTLSHP | R | | CRYSTAL SHAPE PARAMETER |
| 2F52 | 3152 | 3152 | 3120 | 3120 | CMMDEX | T | | TD FOR CMMDEX |
| 2F7A | 317A | 317A | 3148 | 3148 | MEASDO | T | | TD FOR MEASDO |
| 2FA2 | 31A2 | 31A2 | 3170 | 317D | CMFINP | T | | TD FOR CMFINP |
| 2FCA | 31CA | 31CA | 3198 | 3198 | CMFOUT | T | | TD FOR CMFOUT |
| 2FF2 | 31F2 | 31F2 | .... | .... | DISK00 | T | | TD FOR DISK00 |
| 3006 | 3206 | 32D6 | .... | .... | DISK01 | T | | TD FOR DISK01 |
| .... | .... | .... | 31C0 | 31CD | DSKOUT | T | | TD FOR DSKOUT |
| 302E | 322E | 322E | 31E8 | 31E8 | DIACNT | T | | TD FOR DIACNT |
| 3056 | 3256 | 3256 | 3210 | 3210 | ANACNT | T | | TD FOR ANACNT |
| 3187 | 3387 | 3387 | 3341 | 3341 | ALARMF | I1 | | ALARM TIMER INTERRUPT FLAG |
| 3188 | 3388 | 3388 | 3342 | 3342 | TIME | I2 | | SYSTEM TIME (INTEGER) |
| 318A | 338A | 338A | 3344 | 3344 | DIFFTM | I2 | | DIFFERENTIAL TIME FOR MACRO EXECUTION |
| 31A4 | 33A4 | 33A4 | 335E | 335E | DTINTV | I1 | | DATA FILE UPDATING INTERVAL |
| 31A6 | 33A6 | 33A6 | 336D | 336D | TIMSET | I2 | | SETPOINT FOR ALARM TIMER (MACRO EXE.) |
| 3243 | 3443 | 3443 | 33FD | 33FD | IOFLAG | I1 | 4 | I/O FLAG ARRAY |
| 32FD | 34FD | 3581 | 353B | 353B | ITEMP1 | I2 | | MEASURED DATA (INTEGER): TEMPERATURE |
| 32FF | 34FF | 3583 | 353D | 353D | ITEMP2 | I2 | | |
| 33D1 | 3501 | 3585 | 353F | 353F | ITEMP3 | I2 | | |
| 3303 | 3503 | 3587 | 3541 | 3541 | ISEEDL | I2 | | SEED LIFT SPEED |
| 33D5 | 3505 | 3589 | 3543 | 3543 | ICRUCL | I2 | | CRUCIBLE LIFT SPEED |
| 33D7 | 35D7 | 358B | 3545 | 3545 | ISEEDR | I2 | | SEED ROTATION |
| 3309 | 3509 | 358D | 3547 | 3547 | ICRUCR | I2 | | CRUCIBLE ROTATION |
| 330B | 350B | 358F | 3549 | 3549 | IPOUT1 | I2 | | POWER OUTPUT (FROM SCR) |
| 330D | 350D | 3591 | 354B | 354B | IPOUT2 | I2 | | |

| V1.6 | V2.0 | V2.1 | V2.2 | V2.3 | | | | |
|---|---|---|---|---|---|---|---|---|
| 330F | 35DF | 3593 | 354D | 354D | IPOUT3 | I2 | | |
| 3311 | 3511 | 3595 | 354F | 354F | IWEIGH | I2 | | WEIGHT |
| 3313 | 3513 | 3597 | 3551 | 3551 | IDWGHT | I2 | | DIFF. WEIGHT |
| 3315 | 3515 | 3599 | 3553 | 3553 | ISEEDP | I2 | | SEED POSITION |
| 3317 | 3517 | 359B | 3555 | 3555 | ICRUCP | I2 | | CRUCIBLE POSITION |
| 3319 | 3519 | 359D | 3557 | 3557 | IBASET | I2 | | BASE TEMPERATURE |
| 331B | 351B | 359F | 3559 | 3559 | IGASPR | I2 | | GAS PRESSURE |
| 331D | 351D | 35A1 | 355B | 355B | CONTAC | I2 | | CONTACT |
| 331F | 351F | 35A3 | 355D | 355D | ANALOG | I2 | 8 | EIGHT SPARE ANALOG CHANNELS (INTEGER) |
| 32FD | 34FD | 3581 | 353B | 353B | ANADAT | I2 | 25 | COMPLETE ARRAY OF ANALOG DATA (INT.) |
| 332F | 352F | 35B3 | 356D | 356D | IDIAMT | I2 | | CRYSTAL DIAMETER (INTEGER) |
| 3331 | 3531 | 35B5 | 356F | 356F | ILENGT | I2 | | LENGTH GROWN (INTEGER) |
| 3333 | 3533 | 35B7 | 3571 | 3571 | SCADIA | R | | SCALING FACTORS: DIAMETER |
| 3337 | 3537 | 35BB | 3575 | 3575 | SCATMP | R | 3 | TEMPERATURES |
| 3343 | 3543 | 35C7 | 3581 | 3581 | SCAMOT | R | 4 | MOTORS |
| 3353 | 3553 | 35D7 | 3591 | 3591 | SCAPWO | R | 3 | POWER OUTPUT |
| 335F | 355F | 35E3 | 359D | 359D | SCAWGT | R | | WEIGHT |
| 3363 | 3563 | 35E7 | 35A1 | 35A1 | SCADWT | R | | DIFFERENTIAL WEIGHT |
| 3367 | 3567 | 35EB | 35A5 | 35A5 | SCAPOS | R | 2 | POSITION |
| 336F | 356F | 35F3 | 35AD | 35AD | SCABST | R | | BASE TEMPERATURE |
| 3373 | 3573 | 35F7 | 35B1 | 35B1 | SCAGAS | R | | GAS PRESSURE |
| 3377 | 3577 | 35FB | 35B5 | 35B5 | SCAPWR | R | | POWER INPUT AND LIMIT |
| 3333 | 3533 | 35B7 | 3571 | 3571 | SCALE | R | 18 | ARRAY OF SCALING FACTORS |
| 337B | 357B | 35FF | 35B9 | 35B9 | CDIASQ | R | | SQUARE OF CRUCIBLE DIAMETER |
| 337F | 357F | 36D3 | 35BD | 35BD | SDIASQ | R | | SQUARE OF SEED DIAMETER |
| 3383 | 3583 | 3607 | 35C1 | 35C1 | OXWGHT | R | | BORIC OXIDE WEIGHT |
| 3387 | 3587 | 360B | 35C5 | 35C5 | RHOXTL | R | | DENSITY: CRYSTAL |
| 338B | 358B | 36DF | 35C9 | 35C9 | RHOMLT | R | | MELT |
| 338F | 358F | 3613 | 35CD | 35CD | RHOOXI | R | | OXIDE |
| 3393 | 3593 | 3617 | 35D1 | 35D1 | SCRUCP | I2 | | SETPOINT FOR CRUC. POSITION (INTEGER) |
| 3395 | 3595 | 3619 | 35D3 | 35D3 | ZEROWT | I2 | | WEIGHT ZEROING VALUE |
| .... | 3597 | 361B | 35D5 | 35D5 | GROWTH | R | | ACTUAL GROWTH RATE |
| 3397 | 359B | .... | .... | .... | RHEIGH | R | | MELT HEIGHT IN CRUCIBLE (REAL) |
| 339B | 359F | .... | .... | .... | INICRP | R | | INITIAL CRUCIBLE POSITION (AT RESET) |
| 339F | 35A3 | .... | .... | .... | INIWGT | R | | INITIAL CRYSTAL WEIGHT (AT RESET) |
| 33A3 | 35A7 | .... | .... | .... | RCRSET | R | | SETPOINT FOR CRUC. POSITION (REAL) |
| 33A7 | 35AB | .... | .... | .... | ADJLEN | R | | LENGTH ADJUSTMENT (REAL) |
| 33AB | 35AF | .... | .... | .... | DIATAB | R | 64 | TABLE OF CRYSTAL DIAMETERS |
| 34AD | 36B1 | .... | .... | .... | DIASTA | I1 | | DIAMETER CONTROLLER STATUS |
| 34B2 | 36B6 | .... | .... | .... | HEIGHT | I2 | | MELT HEIGHT (SCALED AS LENGTH) |
| 34B4 | 36B8 | .... | .... | .... | RHOOXA | R | | ADJUSTED OXIDE DENSITY |
| 34B8 | 36BC | .... | .... | .... | DIA1SQ | R | | SQUARE OF DIAMETER AT OXIDE SURFACE |
| 34BC | 36CD | .... | .... | .... | DIA2SQ | R | | SQUARE OF DIAMETER AT MELT SURFACE |
| 34C2 | 36C6 | .... | .... | .... | POINTS | I2 | | NUMBER OF DATA POINTS IN SUMMATION |
| 34C4 | 36C8 | .... | .... | .... | DIA1SM | R | | SUM OF DIAMETER SQ. AT OXIDE SURFACE |
| 34C8 | 36CC | .... | .... | .... | DIA2SM | R | | SUM OF DIAMETER SQ. AT MELT SURFACE |
| 34D0 | 36D4 | .... | .... | .... | STEP | R | | STEP FOR MELT HEIGHT EVALUATION |
| 34D4 | .... | .... | .... | .... | GROWTH | R | | ACTUAL GROWTH RATE |
| 34D8 | 36D8 | .... | .... | .... | RSEEDL | R | | SEED LIFT SPEED (FLOATING-POINT) |

| V1.6 | V2.0 | V2.1 | V2.2 | V2.3 | | | | |
|---|---|---|---|---|---|---|---|---|
| 34DC | 36DC | ···· | ···· | ···· | RCRUCL | R | | CRUCIBLE LIFT SPEED (FLOATING-POINT) |
| ···· | ···· | 361F | 35D9 | 35D9 | ALPHA | R | | CORRECTION FACTOR FOR GROWTH RATE |
| ···· | ···· | ···· | 35DD | 35DD | DIATAB | R | 64 | DIAMETER SQUARES TABLE |
| ···· | ···· | 3623 | [illegible] | 36DD | IHEIGH | I2 | | MELT HEIGHT (SCALED AS LENGTH) |
| ···· | ···· | 3625 | 36DF | 36DF | OLDLEN | I2 | | LENGTH AT LAST SLICE BOUNDARY |
| ···· | ···· | 3627 | 36E1 | 36E1 | DIFFLG | I2 | | HEIGHT OF CURRENT SLICE |
| ···· | ···· | ···· | ···· | 36E5 | RDWGHT | R | | (ADJUSTED) DIFFERENTIAL WEIGHT |
| ···· | ···· | 362F | 36E9 | 36E9 | RHOOXA | R | | ADJUSTED OXIDE DENSITY |
| ···· | ···· | 3633 | 36ED | 36ED | DIA1SQ | R | | SQUARE OF DIAMETER AT OXIDE SURFACE |
| ···· | ···· | 3637 | 36F1 | 36F1 | DIA2SQ | R | | SQUARE OF DIAMETER AT MELT SURFACE |
| ···· | ···· | 363B | 36F5 | 36F5 | HEIGHT | R | | BORIC OXIDE HEIGHT IN CRUCIBLE (REAL) |
| ···· | ···· | ···· | 36F9 | 36F9 | RCRSET | R | | CRUCIBLE POSITION SETPOINT (REAL) |
| ···· | ···· | 363F | 36FD | 36FD | VOLSUM | R | | SUM OF VOLUMES IN CURRENT SLICE |
| ···· | ···· | 3643 | 3701 | 3701 | OXIVOL | R | | VOLUME OF BORIC OXIDE MELT |
| ···· | ···· | ···· | 3705 | 3705 | CORRVL | R | | OXIDE VOLUME CORRECTION |
| ···· | ···· | ···· | 3709 | 3709 | BETA | R | | CORRECTION FACTOR |
| ···· | ···· | 3647 | 370D | 370D | RDLIFT | R | | SEED - CRUCIBLE LIFT SPEEDS |
| ···· | ···· | 364B | 3711 | 3711 | RLNGTH | R | | UNSCALED LENGTH |
| ···· | ···· | 364F | 3715 | 3715 | PRLNGT | R | | UNSCALED LENGTH DURING PREVIOUS PASS |
| ···· | ···· | 3657 | 371D | 371D | INICRP | R | | CRUCIBLE POSITION AT RESET |
| ···· | ···· | 365B | ···· | ···· | RCRSET | R | | CRUCIBLE POSITION SETPOINT (REAL) |
| ···· | ···· | 365F | 3721 | 3721 | ADJLEN | R | | LENGTH ADJUSTMENT PARAMETER |
| ···· | ···· | 3663 | ···· | ···· | DIATAB | R | 64 | DIAMETER SQUARES TABLE |
| ···· | ···· | 3765 | 3727 | 3727 | DIASTA | I1 | | DIAMETER CONTROLLER STATUS |
| ···· | ···· | 3766 | 3728 | 3728 | LOOPCT | I1 | | LOOP COUNTER LOCATION |
| ···· | ···· | ···· | 3729 | 3729 | OXOVFL | I1 | | OXIDE HEIGHT OVERFLOW FLAG |
| 3EAB | 4164 | 41EB | 4133 | 4179 | MTEMP1 | I2 | | ANALOG MEASUREMENT DATA: TEMPERATURE I |
| 3EAD | 4166 | 41ED | 4135 | 417B | MTEMP2 | I2 | | TEMPERATURE II |
| 3EAF | 4168 | 41EF | 4137 | 417D | MTEMP3 | I2 | | TEMPERATURE III |
| 3EB1 | 416A | 41F1 | 4139 | 417F | MSEEDL | I2 | | SEED LIFT |
| 3EB3 | 416C | 41F3 | 413B | 4181 | MCRUCL | I2 | | CRUCIBLE LIFT |
| 3EB5 | 416E | 41F5 | 413D | 4183 | MSEEDR | I2 | | SEED ROTATION |
| 3EB7 | 4170 | 41F7 | 413F | 4185 | MCRUCR | I2 | | CRUCIBLE ROTATION |
| 3EB9 | 4172 | 41F9 | 4141 | 4187 | MPOUT1 | I2 | | POWER OUTPUT (FROM SCR) |
| 3EBB | 4174 | 41FB | 4143 | 4189 | MPOUT2 | I2 | | |
| 3EBD | 4176 | 41FD | 4145 | 418B | MPOUT3 | I2 | | |
| 3EBF | 4178 | 41FF | 4147 | 418D | MWEIGH | I2 | | WEIGHT |
| 3EC1 | 417A | 4201 | 4149 | 418F | MDWGHT | I2 | | DIFFERENTIAL WEIGHT |
| 3EC3 | 417C | 42D3 | 414B | 4191 | MSEEDP | I2 | | SEED POSITION |
| 3EC5 | 417E | 4205 | 414D | 4193 | MCRUCP | I2 | | CRUCIBLE POSITION |
| 3EC7 | 418D | 42D7 | 414F | 4195 | MBASET | I2 | | BASE TEMPERATURE |
| 3EC9 | 4182 | 42D9 | 4151 | 4197 | MGASPR | I2 | | GAS PRESSURE |
| 3ECB | 4184 | 420B | 4153 | 4199 | MCONTC | I2 | | CONTACT DEVICE |
| 3EA9 | 4162 | 41E9 | 4131 | 4177 | MANALG | I2 | 25 | ANALOG DATA INPUT ARRAY |

## APPENDIX G: DISK ERROR CODES

RXISIS-II and the CGCS return a numeric error code in the case of a disk error. The error codes are the same in either environment. Although some error messages are trapped by application programs (under RXISIS-II) or by the CGCS, and replaced by more detailed message text, many errors are displayed by the generic error message generation routines which provide the error code only, without an explanation. The CGCS returns, in addition to the plain error code, a message

```
***** DISK ERROR xxx yy (TASK tsknam, LOC hexl) *****
```

which is accompanied by a "beep". In the above message, "xxx" is replaced by the major, and "yy", by the minor error codes; "tsknam" stands for the name of the task which detected the error, and "hexl" represents the absolute program code address where the error was recognized.

The task name displayed with the disk error message indicates which CGCS file was involved in the error:

General System Operations:
- RXIROM - Overlay or auxiliary file handling.

Macro Command File:
- RXIROM - (Conditional) Macro call from console.
- CMMDEX - (Conditional) Macro call from console or Macro file.
- CMFINP - Macro command execution.

Print File:
- RXIROM - At all times.
- CMMDEX - (Error) message output.
- CMFINP - (Error) message output.
- DIACNT - (Error) message output.

Data File:
- RXIROM - During opening and closing and upon a COMMENT command.
- DSKOUT - During regular operation.

Control Output File:
- RXIROM - During opening and closing.
- CMFOUT - During regular operation.

## Appendix G: Disk Error Codes

The following error codes are returned by RXISIS-II and by the CGCS:

| | |
|---|---|
| 2 | Invalid file number |
| 3 | Attempt to open more than 6 files simultaneously *) |
| 4 | Illegal file name |
| 5 | Illegal device name |
| 6 | Attempt to write to a file open for input |
| 7 | Disk is full |
| 8 | Attempt to read from a file open for output |
| 9 | Disk directory is full |
| 10 | Different disks in RENAME call |
| 11 | File name is already in use |
| 12 | File is already open |
| 13 | No such file |
| 14 | Attempt to write to a write protected file |
| 15 | Attempt to load into protected memory area |
| 16 | Incorrect object program format |
| 17 | Attempt to access a non-disk file |
| 18 | Unrecognized message type or system call |
| 19 | Attempt to seek on a non-disk file |
| 20 | Attempt to seek in front of beginning of a file |
| 22 | Illegal access parameter in OPEN call |
| 24 | Disk I/O (hardware) error |
| 26 | Illegal attribute parameter in ATTRIB call |
| 27 | Illegal mode parameter in SEEK call |
| 28 | Missing file name extension *) |
| 29 | End of console file |
| 30 | Disk drive not ready |
| 31 | Attempt to seek on a file open for output |
| 32 | Attempt to delete an open file |
| 35 | Attempt to seek past end of file open for input |
| 40 | Request sent to wrong exchange |
| 41 | Insufficient free memory to open file |
| 42 | Drive not in configuration table |
| 43 | Drive timeout |
| 120 | Insufficient memory to open new file #) |
| 121 | Attempt to load a main program |
| 218 | Unallocated disk file block prior to EOF |

*) RXISIS-II only
#) CGCS only

Minor error code information is only displayed in case of an error 24 (Disk I/O error):

| | |
|---|---|
| 01 | Deleted record |
| 02 | Cyclic redundancy check error (data field) |
| 03 | Invalid address mark |
| 04 | Seek error |

| | |
|---|---|
| 08 | Address error |
| 0A | Cyclic redundancy check error (ID field) |
| 0E | No address mark |
| 0F | Incorrect data address mark |
| 10 | Data overrun or underrun |
| 20 | Disk is write protected |
| 40 | Write error |
| 80 | Not ready |

# APPENDIX H: CGCS FILE FORMATS

## H 1: VARIABLE NAME FILE CZONAM.Vmn

The file name extension of the Variable Name file has to hold the major and minor version codes of the CGCS system release to which the file refers. (CZONAM.V23 is, for example, the Variable Name file for CGCS Version 2.3.) The file is built of records of 128 bytes, each of which holds 14 entries of 9 bytes each; each record is terminated by a Carriage Return - Line Feed pair.

Each entry contains:

| | |
|---|---|
| Bytes 1 - 6: | Variable name (1 - 6 uppercase characters, left justified, right filled with spaces. |
| Byte 7: | Variable type and array size, encoded as (type number + (array size - 1) * 4), where "array size" is the number of array elements (1 to 64). The following "type" values are defined:<br><br>type = 0 ... iRMX-80 control structure<br>type = 1 ... one-byte integer (INTEGER*1)<br>type = 2 ... two-byte integer (INTEGER*2)<br>type = 3 ... floating-point number (REAL) |
| Bytes 8 - 9: | Start address of the specified variable. |

## H 2: VARIABLE NAME SOURCE FILE

The source file which holds the variable names and which is eventually converted to a CZONAM file with the utility program CONVAD does not require very strict formatting but must follow the subsequent rules:

(1) Each entry must be held in a separate line in the following order:

(a) Address (in hexadecimal notation, with or without trailing "H").

(b) Variable name (in capitals), 1 to 6 characters.

(c) Variable type number (0 through 3; see chapter H 1).

(d) Number of array elements (optionally); a missing number is interpreted as "1".

(e) Comment (optional); the comment field should not contain digits lest they could be interpreted as an array size.

(2) Entries must be separated from one another by one or more blanks (spaces, TAB characters, etc.).

## H 3: MACRO COMMAND FILES

Macro files (and therefore also the Command Output files) are built of records of 16 bytes each. They consist of one header record and an arbitrary number (including zero) of data records.

### Header Record:

| | |
|---|---|
| Bytes 1 - 2: | Zero. |
| Byte 3: | Minor CGCS Version code. |
| Byte 4: | Major CGCS Version code. |
| Bytes 5 - 16: | Don't care. |

### Data Records:

| | |
|---|---|
| Bytes 1 - 2: | Relative time of command as unsigned two-byte integer (0 - 65535). |
| Byte 3: | Command code byte. |
| Bytes 4 - 16: | Depend on command code; see below. |

### Command Codes:

| | |
|---|---|
| 11H | Set Diameter |
| 12H | Set Heater Temperature #1 |
| 13H | Set Heater Temperature #2 |
| 14H | Set Heater Temperature #3 |
| 15H | Set Seed Lift speed |
| 16H | Set Crucible Lift speed |
| 17H | Set Seed Rotation speed |
| 18H | Set Crucible Rotation speed |
| 19H | Set Power Limit |
| 21H | Modify Diameter |
| 22H | Modify Heater Temperature #2 |
| 23H | Modify Heater Temperature #2 |
| 24H | Modify Heater Temperature #3 |

```
25H  Modify Seed Lift speed
26H  Modify Crucible Lift speed
27H  Modify Seed Rotation speed
28H  Modify Crucible Rotation speed
29H  Modify Power Limit
     Bytes 4 - 5:   New setpoint or setpoint change
                    (INTEGER*2).
     Bytes 6 - 9:   Transition time in seconds (REAL).
     Bytes 10 - 16: Don't care.

30H  Macro Command
     Bytes 4 - 9:   Macro Command name (left justified,
                    right filled with spaces).
     Bytes 10 - 16: Don't care.

31H  Clear Conditional Macros Unconditionally
     Bytes 4 - 16:  Don't care.

40H  Mode = Monitoring
41H  Mode = Manual
42H  Mode = Diameter
43H  Mode = Diameter/ASC
44H  Mode = Automatic
     Bytes 4 - 16:  Don't care.

70H  Reset
     Bytes 4 - 5:   New weight (INTEGER*2).
     Bytes 6 - 7:   New length (INTEGER*2).
     Bytes 8 - 16:  Don't care.

7FH  End of Command Record
     Bytes 4 - 16:  Don't care.

90H  Set Variable
A0H  Change Variable
     Byte 4:        Variable type:
                          2 ... INTEGER*1
                          4 ... INTEGER*2
                          6 ... REAL
     Bytes 5 - 6:   Variable address (INTEGER*2).
     Bytes 7 - 10:  New setpoint or change value (REAL).
     Bytes 11 - 14: Transition time in seconds (REAL).
     Bytes 15 - 16: Don't care.

B0H  Conditional Command
     Byte 4:        Variable type + 16 * Relation code #2
                    + 64 * Relation code #1, with:
                    Variable type:
                          2 ... INTEGER*1
                          4 ... INTEGER*2
```

```
                        6 ... REAL
                  Relation code:
                        1 ... "<"
                        2 ... "="
                        3 ... ">"
     Bytes 5 - 6:   Variable address (INTEGER*2).
     Bytes 7 - 10:  Comparison value (REAL).
     Bytes 11 - 16: Macro Command name.

B1H  Clear Conditional Commands Selectively
     Byte 4:        Don't care.
     Bytes 5 - 6:   Variable address (INTEGER*2).
     Bytes 7 - 16:  Don't care.

E0H  Assign Plot Channel
     Byte 4:        Plot channel number (1 - 8)
     Bytes 5 - 6:   Variable address (INTEGER*2).
     Bytes 7 - 16:  Don't care.

F2H  Debug Continuously
F3H  Debug Modify
F4H  Debug Resume
F5H  Debug Suspend
     Byte 4:        Variable type + 16 * Output location,
                    with:
                    Variable type:
                          1 ... ASCII (1 character)
                          2 ... INTEGER*1
                          3 ... one-byte hexadecimal
                          4 ... INTEGER*2
                          5 ... two-byte hexadecimal
                          6 ... REAL
                          7 ... four-byte hexadecimal
                    Output location: 1 - 4
     Bytes 5 - 6:   Variable address (INTEGER*2).
     Bytes 7 - 10:  New value.
     Bytes 11 - 16: Don't care.
     (Most Debug commands need only part of the informa-
     tion in bytes 4 - 10)
```

The contents of a command message are identical to those of the corresponding command record bytes 3 through 16.

## H 4: DATA FILES

A Data file is made up of records of 128 bytes each. It consists of one Header record and an arbitrary number of Data and Comment records. With the exception of the first two bytes, Data records are built of two-byte words, i.e., 64 words per record. All data is in INTEGER*2 format unless noted otherwise.

Header Record:

| | |
|---|---|
| Bytes 1 - 8: | Date (8 ASCII characters). |
| Bytes 9 - 28: | Run Identification (20 ASCII characters). |
| Bytes 29 - 30: | Record interval (two hexadecimal digits). |
| Byte 31: | Major CGCS system version code. |
| Byte 32: | Minor CGCS system version code. |
| Bytes 33 - 128: | Contents of bytes 1 - 32 repeated three times. |

Data Record:

| | |
|---|---|
| Byte 1: | Always 0. |
| Byte 2: | Operation Mode (INTEGER*1). |
| Word 2: | System time. |
| Word 3: | Length grown. |
| Word 4: | Temperature #1 (Measured Data). |
| Word 5: | Temperature #2. |
| Word 6: | Temperature #3. |
| Word 7: | Seed Lift. |
| Word 8: | Crucible Lift. |
| Word 9: | Seed Rotation. |
| Word 10: | Crucible Rotation. |
| Word 11: | Power Output #1. |
| Word 12: | Power Output #2. |
| Word 13: | Power Output #3. |
| Word 14: | Weight. |
| Word 15: | Differential Weight. |
| Word 16: | Seed Position. |
| Word 17: | Crucible Position. |
| Word 18: | Base Temperature. |
| Word 19: | Gas Pressure. |
| Word 20: | Contact Device. |
| Word 21 - 28: | Eight Spare Analog Input Channels. |
| Word 29: | Power Input #1 (Control Output). |
| Word 30: | Power Input #2. |
| Word 31: | Power Input #3. |

| | |
|---|---|
| Word 32: | Diameter (Current Setpoints). |
| Word 33: | Temperature #1. |
| Word 34: | Temperature #2. |
| Word 35: | Temperature #3. |
| Word 36: | Seed Lift. |
| Word 37: | Crucible Lift. |
| Word 38: | Seed Rotation. |
| Word 39: | Crucible Rotation. |
| Word 40: | Power Limit. |
| Word 41: | Diameter (Final Setpoints). |
| Word 42: | Temperature #1. |
| Word 43: | Temperature #2. |
| Word 44: | Temperature #3. |
| Word 45: | Seed Lift. |
| Word 46: | Crucible Lift. |
| Word 47: | Seed Rotation. |
| Word 48: | Crucible Rotation. |
| Word 49: | Power Limit. |
| Word 50: | Debug Continuously Address #1. |
| Word 51 - 52: | Debug Continuously Data #1 (4 bytes). |
| Word 53: | Debug Continuously Address #2. |
| Word 54 - 55: | Debug Continuously Data #2 (4 bytes). |
| Word 56: | Debug Continuously Address #3. |
| Word 57 - 58: | Debug Continuously Data #3 (4 bytes). |
| Word 59: | Debug Continuously Address #4. |
| Word 60 - 61: | Debug Continuously Data #4 (4 bytes). |
| Word 62: | Diameter (Calculated Value). |
| Word 63: | Spare. |
| Word 64: | Debug Continuously Type Flags (compare chapter H 3, Debug Variable types:) TYPE(1) + 16*TYPE(2) + 256*TYPE(3) + 4096*TYPE(4) |

Comment Records:

|  |  |
|---|---|
| Byte 1: | Always -1. |
| Bytes 2 - 6: | as in Data Records. |
| Bytes 7 - 128: | Comment input (122 ASCII characters; only the first 79 are displayed by SHODAT). |

## APPENDIX I: CZOCHRALSKI GROWTH CONTROL SYSTEM MESSAGES

In addition to immediate responses to operator commands, the CGCS may issue messages to the console and to a Documentation output (if available) which need not obviously be triggered by operator entries. For reasons of brevity, only the messages which are not generated by the Command Interpreter are listed below in alphabetical order. (The Command Interpreter responses are self-explanatory and always immediately related to an operator entry.) In general, messages starting with "*****" have informational character only, whereas "#####" may indicate a genuine error condition. The latter messages are, in general, accompanied by a "beep". (Exceptions to this rule are the Disk, Input, Output, Printer, and System error messages which are tagged with asterisks. They are generated by the operating system and are displayed only on the console.)

***** All Conditional Macros cleared *****

An Unconditional CLEAR command (i.e., a CLEAR command without any parameter) was entered from the console or from a Macro Command file.

***** Automatic RESET executed - automatic Mode changes will follow *****

The operation mode was changed into a Diameter controlled mode while the Diameter Evaluation routines were not yet initialized with a RESET command. The system takes care of that on its own in a somewhat complicated procedure.

##### Can't calculate diameter with zero seed lift speed

The actual seed lift speed is still zero when RESET is commanded, or it is set to zero while the Diameter Evaluation routines are active.

##### Can't control system

The operator or a Macro Command attempted to SET or CHANGE a parameter or Variable while in Monitoring mode. The command is executed, though, but it may become ineffective in the case of a change to any controlled mode.

## Appendix I: Czochralski Growth Control System Messages

##### Can't ramp parameter

The maximum number of parameters or Variables (20) were already being ramped when a SET or CHANGE command with non-zero transition time was issued. The change is effected immediately, without ramping.

##### Command Macro call ignored

A specified Macro was not found, or a disk error occurred while the Macro file header was read.

***** Command Macro preempted *****

A Macro Command was activated, either from a pending Conditional command, or through an unconditional Macro Command, while another Macro was active.

***** Conditional Macro cleared *****

A Selective CLEAR command has removed one Conditional Macro from the Conditional Command queue. This message is repeated for each Conditional Command cancelled with a Selective CLEAR; it may therefore appear multiplely.

##### Conditional Macro Command ignored

A Conditional Macro Command was encountered while already the maximum number of Macro Commands (8) were pending.

***** Conditional Macro started *****

A Conditional Macro Command has met its condition and is activated.

##### Continued speed overflow - RESET required

The system cannot automatically recover from a serious problem.

##### Crystal shape adjusted

The calculated diameter value changed faster than permitted. The diameter value stored in memory for the diameter and crucible position evaluation is corrected to differ from the value stored before exactly by the permitted maximum. Crystal shape adjustments may cause minor transients in the calculated diameter and/or crucible position setpoint.

***** DISK ERROR xxx yy (TASK tsknam, LOC hexl) *****

Disk error message provided by the operating system.

***** End of Macro command file *****

The end of a Macro file was reached, or a disk error prohibited its further execution.

***** Executing Macro MACNAM *****

The Command file with the name MACNAM was started either from an unconditional Macro Command, or from a Conditional Macro Command whose condition was met.

##### Illegal command file format

A Macro Command file has an improper format and cannot be processed.

***** INPUT ERROR *****

Error message generated by the operating system, most likely due to illegal data entry on the console. This message should hardly appear, though.

##### Macro command not executable

A command referring to a Variable or absolute memory location was encountered in a Macro Command file generated for a different CGCS version. The command is ignored.

## Appendix I: Czochralski Growth Control System Messages

##### Macro MACNAM doesn't exist

The Macro Command with the name MACNAM was supposed to be executed either from an unconditional or from a Conditional Macro Command but the file MACNAM.CMD was not found on drive 0. The command is ignored.

##### Meltback detected

The crystal's length was reduced by more than approximately 1 mm since an earlier pass of the Diameter Evaluation routines. The Diameter Evaluation routines continue to operate normally.

##### Mode automatically set to Manual

A zero seed lift speed or a speed overflow error was detected by the Diameter Evaluation routines.

***** New Mode: MODE NAME *****

The CGCS operation mode was set to the mode indicated, either from the operator console, from a Macro Command, or, automatically, in case of a diameter evaluation error.

##### Non-matching Command Macro system version - restricted command set

A Macro Command file generated under or for a different CGCS version was invoked. All commands referring to Variables or absolute hexadecimal addresses will be skipped.

***** OUTPUT ERROR *****

Error message generated by the operating system. This message should never be encountered!

##### Overflow - result limited to permitted maximum

As a result of a SET or CHANGE command, a parameter or Variable would have been set to a value exceeding the permitted range for the particular location.

##### Oxide height overflow - Diameter may be incorrect

The height of the boric oxide melt exceeded the permitted maximum of ca. 75 mm. The maximum melt height is used for diameter and crucible position setpoint evaluation. This data may therefore be incorrect.

##### Parameter can't be negative

A SET or CHANGE command attempted to set a diameter, temperature, or power limit setpoint to a negative value. The setpoint is set to zero instead.

***** PRINTER NOT READY *****

Error message generated by the operating system. The printer was in off-line mode while the system attempted to transfer data to it.

##### PROGRAM CODE DAMAGED AT xxxxH #####

At least one byte within the memory page (= 256 bytes) starting at the address specified in the message was changed since the last pass of the code checker routine, approximately 30 seconds ago. This message should never appear! Preserve all data of the run if it does happen, and report it immediately.

***** Recorder channel N is negative *****
***** Recorder channel N is positive *****

The output to the chart recorder channel N (N is an integer between 1 and 8) changed its sign. Initially, the output data of all channels is supposed to be positive.

***** Regular growth resumed *****

A meltback, zero seed lift speed, or speed overflow condition has been terminated; the Diameter Evaluation routines can resume correct operation.

## Appendix I: Czochralski Growth Control System Messages

##### Speed overflow

The calculated length of the crystal was increased or decreased by more than 2 mm during the last 10 seconds. This may be due to a very fast seed lift, or to an abrupt change of the crystal's weight. The system tries to recover automatically from such a condition.

***** SYSTEM ERROR (TASK tsknam, LOC hexl) *****

This error message should never appear. Preserve all data of the run if it does happen, and report it immediately.

## APPENDIX J: DYNAMIC BEHAVIOR OF THE PID CONTROLLER ROUTINE

Simulations of the PID controller's response were performed for the most important operation modes in order to compare their dynamic response to various shapes of the error signal. For all simulations shown in the subsequent illustrations, the following parameters were used:

| | |
|---|---|
| Proportional Multiplier | P = 256 |
| Integral Multiplier | I = 64 |
| Derivative Multiplier | D = 256 |
| Limit | L = 25 |
| Integral Scaling Factor | IS= 256 |
| Bias | B = 0 |

The setpoint S was kept at 0, and the Actual signal A was set to follow the function depicted in Fig. A1. The first part of this simulation, consisting of two series of 25 passes of FRPIDC with A equal to +10 and -10, respectively, was chosen to represent a small but persistent error for which the proportional (plus derivative) components of the output signal are well below a limit value (if one was chosen). During the next part of the simulation, A was increased to ±20 units for 5 passes each; for the ensuing error, the limit is to be incurred essentially due to the proportional and derivative components. The simulation is concluded with two single-pass pulses of A with a magnitude of ±50 units which were provided to represent the behavior of the controller for large transients.

In Fig. A2, the controller's response is shown for a CNTL value of 0, i.e., for no limiting and anti-windup operation. Note that, according to eq. (1), the sign of the controller's output is opposite to the sign of the input value A. (This approach results in positive controller parameters for most applications.) During the first part of the simulation, the response of the controller is essentially determined by the integral component; the integral and derivative components are only superimposed. Note that it takes a long time after the error reversed its sign until the controller's output signal (full line) changes its sign. The dynamic response is improved during part 2 of the simulation since the proportional and derivative components dominate there. The concluding single error pulses result in a strong output signal in the proper direction, followed by the opposite overshoot caused by the reaction of the derivative component to the trailing edge of the pulse. Since the controller is linear and the input signal symmetrical, the error integral returns to zero after each part of the test.

This is also true for the second set of control flags tested, namely, for CNTL equal to 2 (Fig. A3). In this case, the controller's output is limited to ±25, but aside from this limiting, the controller is still linear. The major drawback of simple output limiting can be seen in the first part of the controller's response curve, where there is no indication in the output signal that the error went to zero, and eventually changed its sign. (Note that, due to internal programming reasons, the output signal is limited to -26 units rather than -25. This fact is very unlikely to matter in actual applications, though, considering an output signal range of the controller of ±32767 units.) Similarly, the controller goes into saturation immediately at the beginning of the second part of the test, and remains there, although the error drops back to zero, aside from a short spike caused by the derivative component. It requires a considerable error with the opposite sign to obtain an output signal with the expected direction. Output limiting also strongly affects the response to large transients: The controller's output bounces back and forth between its negative and positive limits. Since transients of this kind are most likely artifacts which better should not be regarded by the controller at all, output signal limiting obviously contributes to a suppression of these short pulses; the positive and negative spikes will cancel their effects mutually in most applications.

In order to improve the dynamic response, particularly, to long-term error conditions where the integral component predominates, the anti-windup function Mode A was provided in FRPIDC. Fig. A4 shows the response of the controller with this feature activated in addition to output signal limiting (CNTL = 6). Indeed, the transition of A from 10 to 0 units has a clear influence upon the output signal, and a response with the expected sign is almost immediately obtained when A changes from 0 to -10 units. There is also a reasonable response to the larger error pulses in the second part of the test; in fact, the response is very similar to the one obtained for no output limit, but, for anti-windup Mode A, the output signal is better centered around zero independent from the preceding history of the controller. However, Mode A fails to function totally for the large transients of the third part of the test. While there is only a small effect of a transient on the steady-state signal <u>after</u> the transient in the operation modes discussed above, Mode A sets the error integral to a large value whose sign depends on the relative magnitude of the proportional and the derivative multipliers and the previous "history" of the controller; in fact, any output value between the positive and the negative maximum may ensue after a larger transient.

In order to counteract this not very desirable behavior, anti-windup Mode B was developed where the error integral is not set to a value depending on the proportional and derivative components if the output exceeds the limit as it was in Mode A; in contrast, the error integral is clamped to the positive or negative limit value, depending on the sign of the total controller output. The response in Mode B (CNTL = 14) is shown in Fig. A5. There is no big difference between Modes A and B for small long-term errors, although Mode B reacts more slowly to changes of the controller's input than Mode A does. The end of the first part of the test sequence, and, even more pronounced, the end of the second part shows, however, the main drawback of this mode: The error integral tends to "get stuck" at either of the controller's limits, and positive action (i.e., an input signal which eventually will reverse the error integral's sign) is required to remove it from there. Furthermore, some anomalies may also happen in Mode B when large error transients are encountered. More or less the expected result is returned for the first spike: The output signal (and the error integral) bounces from the positive to the negative limit, and returns to the positive limit. The treatment of the second spike is less obvious. There is no visible response to the leading slope of the spike because the positive output which would have resulted from it is clipped off by the limiting operation. During the trailing slope, however, the derivative component determines the output signal. Incidentally, an output signal resulted in our simulations which was next to the negative limit but, from the controller's point of view, not beyond the limit. The integral component was therefore not modified but remained at its positive limit. (Had we used a pulse amplitude of 51 rather than 50 units, we would have obtained a reversal of the integral component's sign.)

The remaining two simulations (Figs. 6 and 7) were based upon a different approach: Rather than modifying the error integral when the <u>output</u> signal exceeds a limit, the error integral <u>itself</u> is kept within the bounds of a limit, no matter what the output signal looks like. This error integral limiting may be used with or without limiting of the final controller output. (The same limit value must be used, though, in both cases.) A simulation without output limiting (CNTL = 16) is shown in Fig. A6, while Fig. A7 shows the effects of output limiting (CNTL = 18). Again the behavior of the controller suffers from the nonlinear response of the error integral which does not return to zero when an error condition occurred although the input signal is symmetric. (It is questionable, however, how representative the test signals chosen here are with respect to actual operating conditions.) Indeed, the response shown in Fig. A7 for integral <u>and</u> output limiting is very similar to the one obtained with anti-windup Mode B in Fig. A5. The only differences occur for the handling of the large transients in the third part of the test. In this case, integral limiting seems to render more consistent and reasonable results, compared to the anti-windup schemes.

Changing the integral scaling factor IS from 256 to 65536 has no effect on the behavior of the controller except that the integral reacts by a factor of 256 slower. Setting IS to 65536 and I to 16384 (64*256) resulted in exactly the same responses as discussed above.

Note that the PID controller routine checks for an output signal overflow after it calculated (and possibly limited) the error integral. In general, there is no point to keep integral limiting and any of the anti-windup schemes active at the same time because the anti-windup algorithms will override (and overwrite) the results of the integral limiter, except for some extremely weird operating conditions where an error reduces its magnitude at a rate fast enough to have the proportional component of the controller output overcompensated by the derivative component, without the error changing its sign. In this case, limiting of the error integral might occur without the output signal exceeding the limit. For obvious reasons, this case was not investigated; for all practical purposes, operation modes 20 through 23 and 28 through 31 are identical to the corresponding modes 4 through 7 and 12 through 15.

## APPENDIX K: DOCUMENTATION PRINTING HISTORY

| | |
|---|---|
| Issue 1: | **January 1986** (vii + 133 pages) |
| | Valid for CGCS versions up to 1.5. |
| Issue 2: | **July 1986** (vii + 155 pages) |
| | Valid for CGCS versions up to 2.0. |
| Issue 3: | **December 1986** (xi + 182 pages) |
| | Valid for CGCS versions up to 2.3. |

## ILLUSTRATIONS

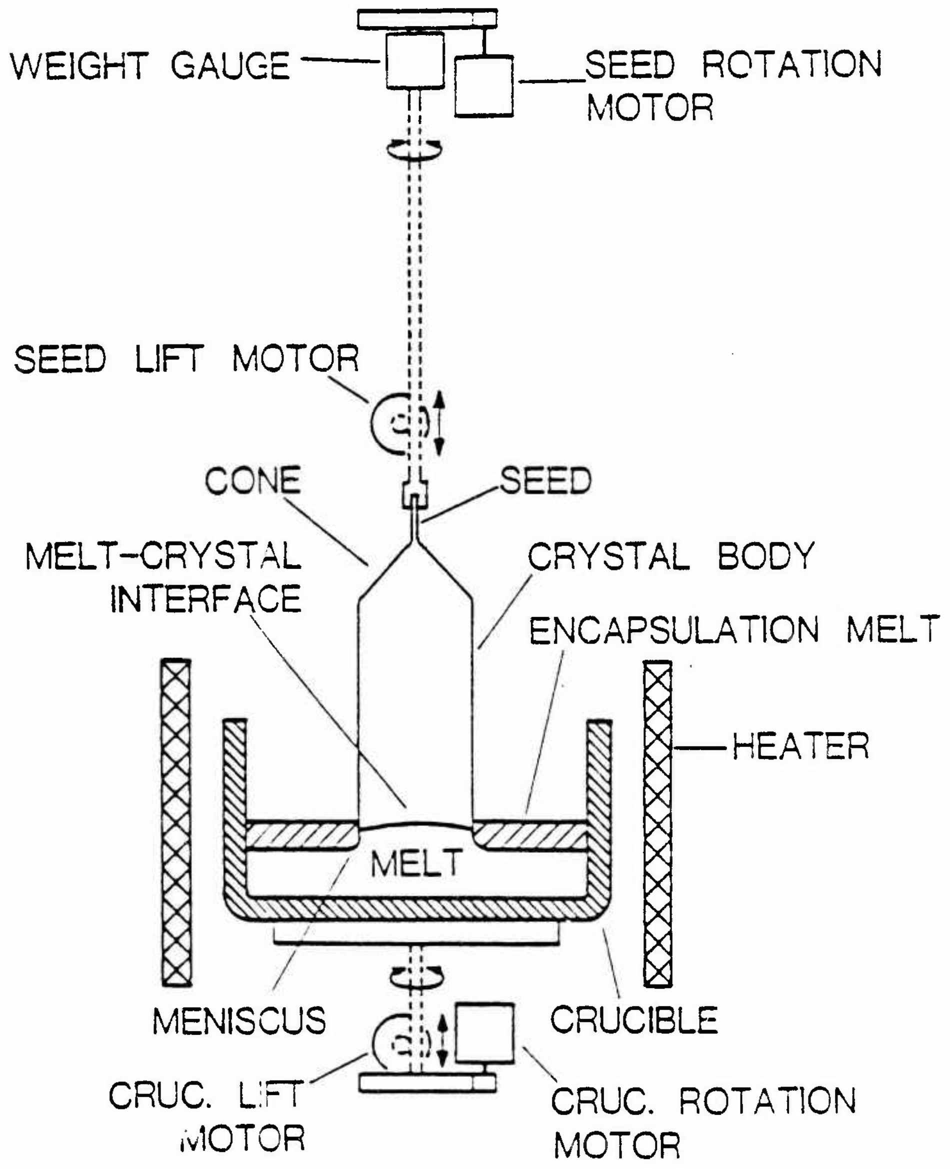


Fig. 1: A Czochralski Puller for Compound Semiconductor Crystal Growth.

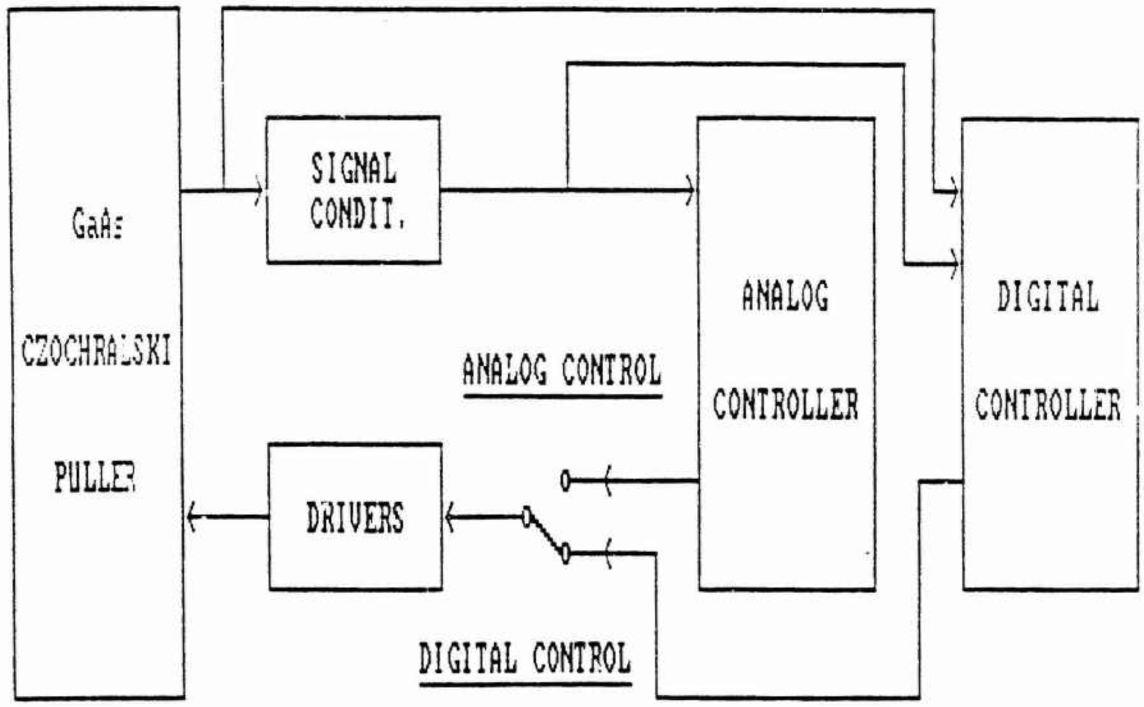


Fig. 2: Implementation of the Digital Czochralski Growth Control System.

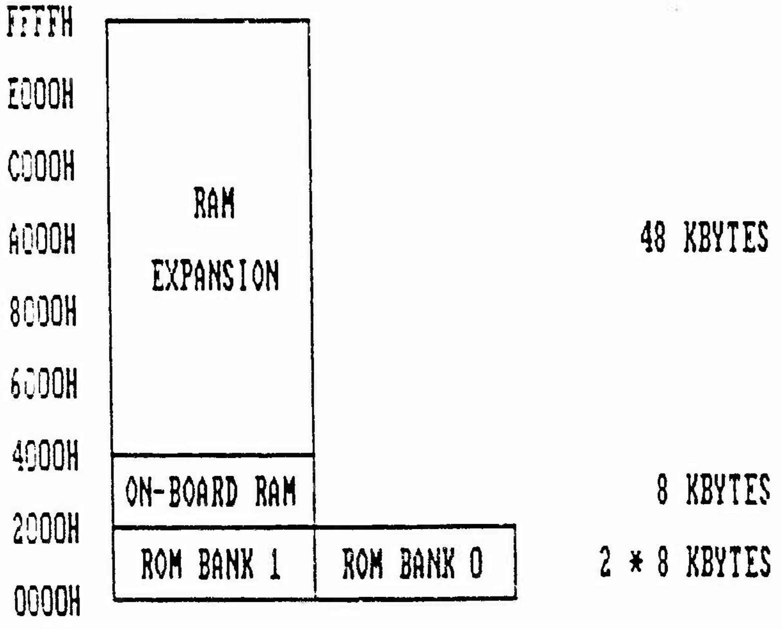


Fig. 3: Hardware Memory Map of the CGCS Computer.

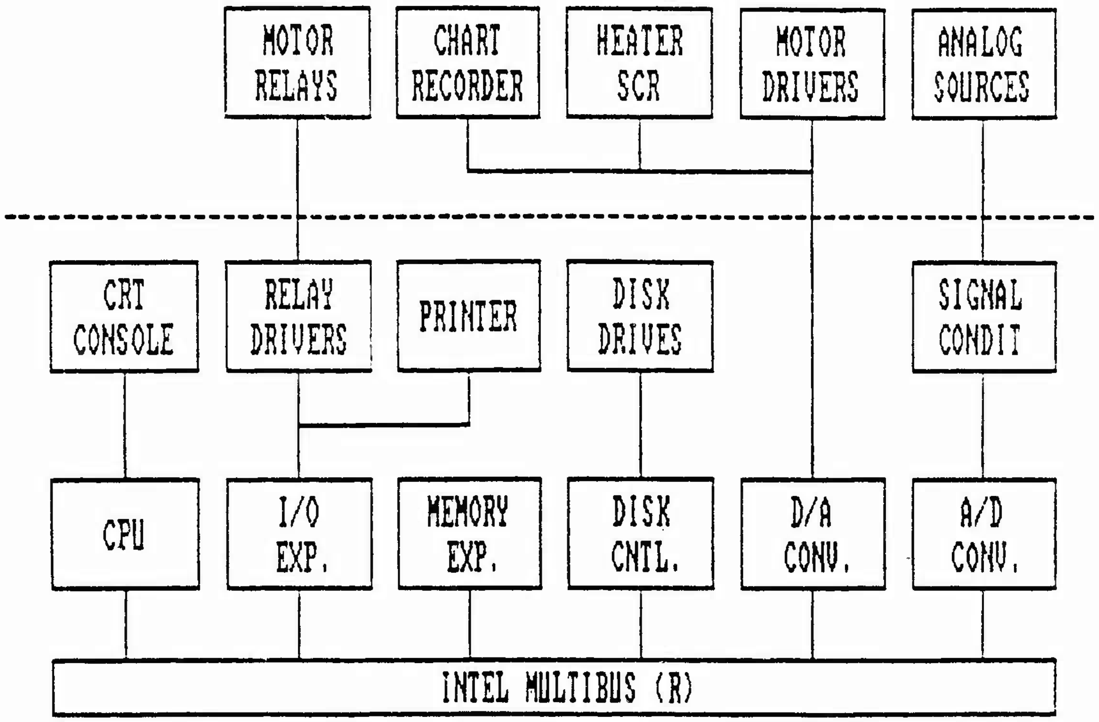


Fig. 4: Hardware Configuration of the CGCS Computer.

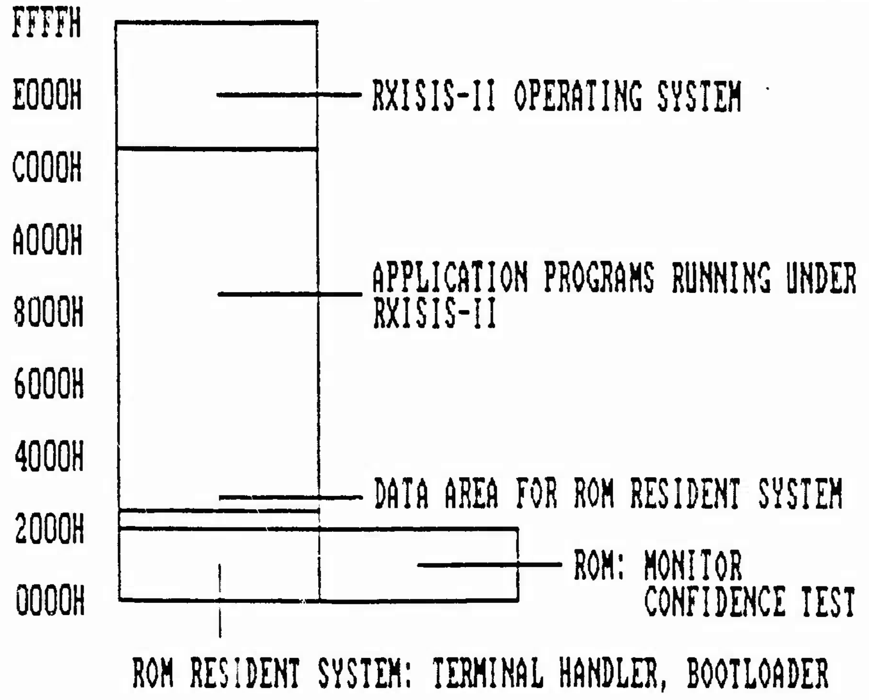

FFFFH
E000H
C000H
A000H
8000H
6000H
4000H
2000H
0000H
RXISIS-II OPERATING SYSTEM
APPLICATION PROGRAMS RUNNING UNDER RXISIS-II
DATA AREA FOR ROM RESIDENT SYSTEM
ROM: MONITOR CONFIDENCE TEST
ROM RESIDENT SYSTEM: TERMINAL HANDLER, BOOTLOADER

Fig. 5: Memory Map of the Computer Under RXISIS-II.

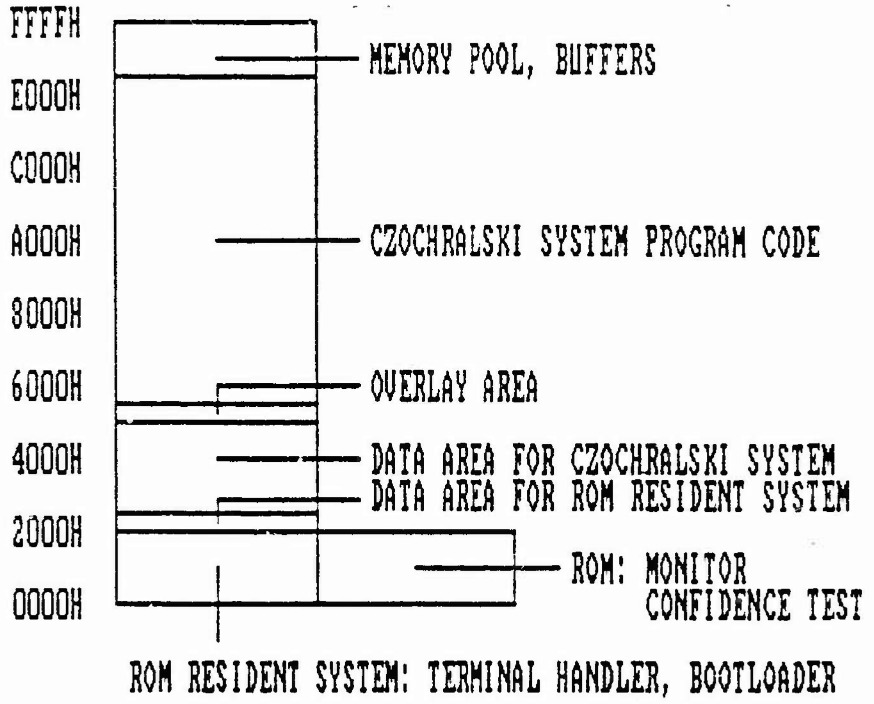

FFFFH
E000H
C000H
A000H
8000H
6000H
4000H
2000H
0000H
MEMORY POOL, BUFFERS
CZOCHRALSKI SYSTEM PROGRAM CODE
OVERLAY AREA
DATA AREA FOR CZOCHRALSKI SYSTEM
DATA AREA FOR ROM RESIDENT SYSTEM
ROM: MONITOR
CONFIDENCE TEST
ROM RESIDENT SYSTEM: TERMINAL HANDLER, BOOTLOADER

Fig. 6: Memory Map of the Computer Under the CGCS.

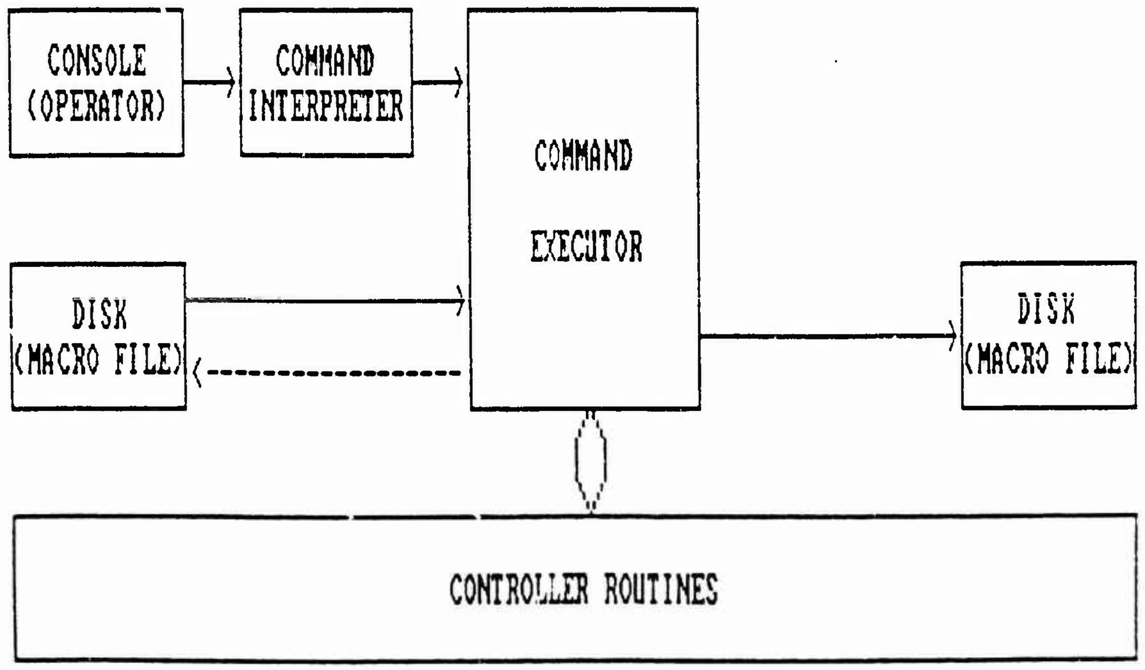


Fig. 7: Function Blocks of the CGCS.

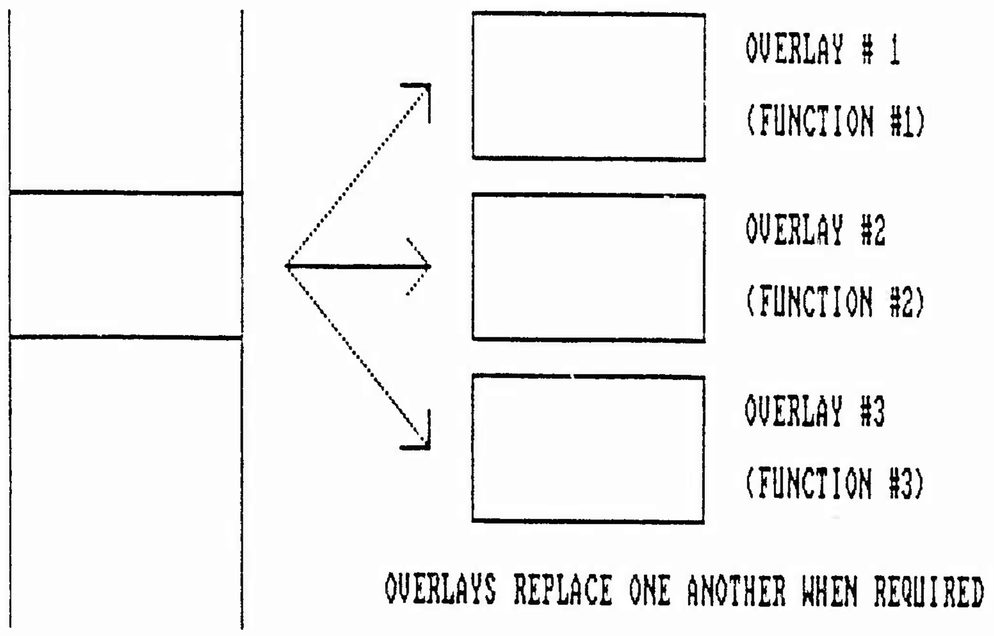
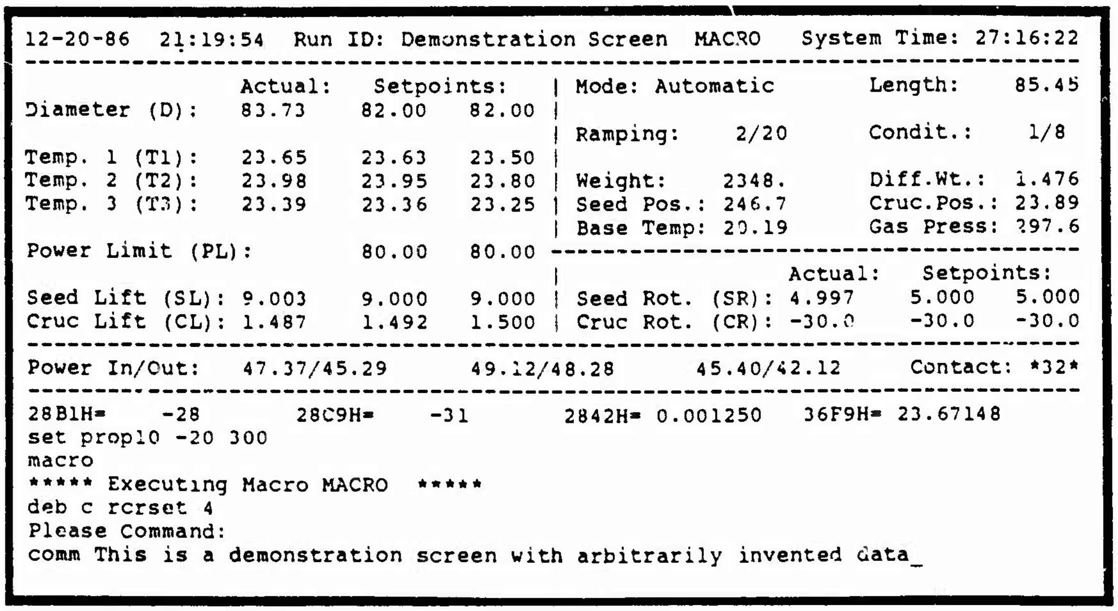


Fig. 8: Program Overlays.



```
12-20-86  21:19:54  Run ID: Demonstration Screen  MACRO   System Time: 27:16:22
-------------------------------------------------------------------------------
                    Actual:   Setpoints:     | Mode: Automatic       Length:    85.45
Diameter (D):       83.73     82.00    82.00 |
                                             | Ramping:    2/20      Condit.:    1/8
Temp. 1 (T1):       23.65     23.63    23.50 |
Temp. 2 (T2):       23.98     23.95    23.80 | Weight:     2348.     Diff.Wt.:  1.476
Temp. 3 (T3):       23.39     23.36    23.25 | Seed Pos.: 246.7      Cruc.Pos.: 23.89
                                             | Base Temp: 20.19      Gas Press: 297.6
Power Limit (PL):             80.00    80.00 ---------------------------------
                                             |             Actual:   Setpoints:
Seed Lift (SL): 9.003         9.000    9.000 | Seed Rot. (SR): 4.997    5.000   5.000
Cruc Lift (CL): 1.487         1.492    1.500 | Cruc Rot. (CR): -30.0    -30.0   -30.0
-------------------------------------------------------------------------------
Power In/Out:   47.37/45.29      49.12/48.28      45.40/42.12      Contact: *32*
-------------------------------------------------------------------------------
28B1H=    -28         28C9H=    -31         2842H= 0.001250    36F9H= 23.67148
set prop10 -20 300
macro
***** Executing Macro MACRO  *****
deb c rcrset 4
Please Command:
comm This is a demonstration screen with arbitrarily invented data_
```

Fig. 9: Console Screen of the CGCS.

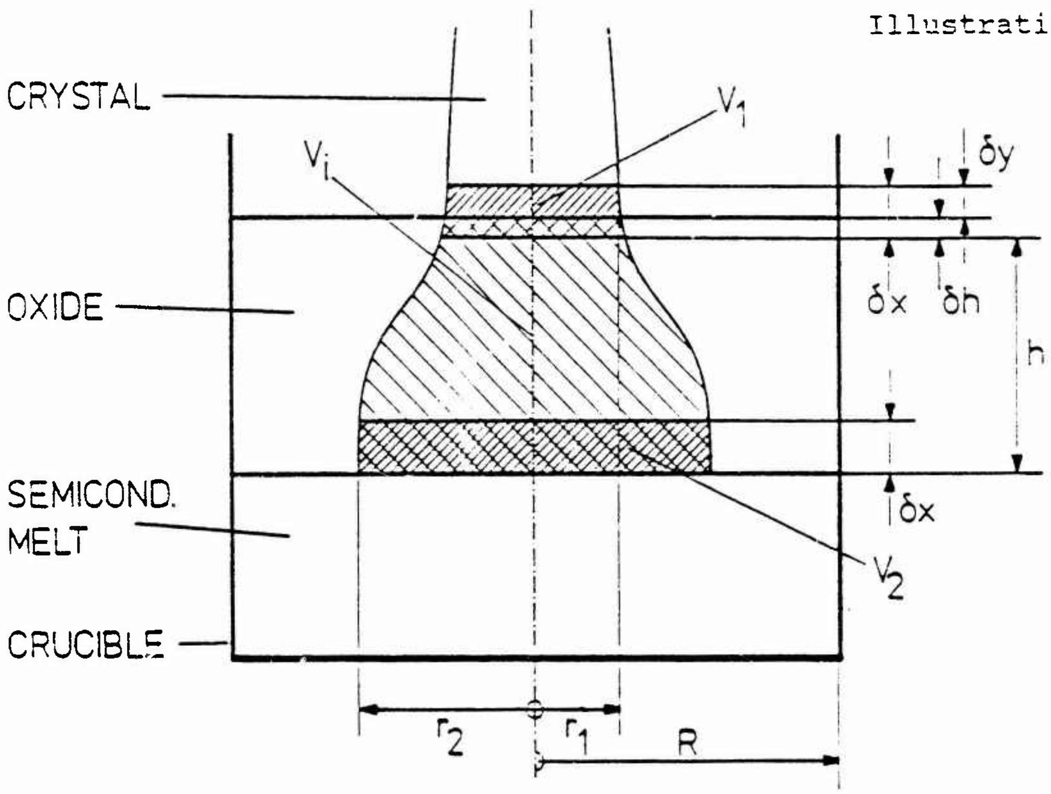


Fig. 10: Growth of a Crystal Partially Immersed in an Oxide Melt.

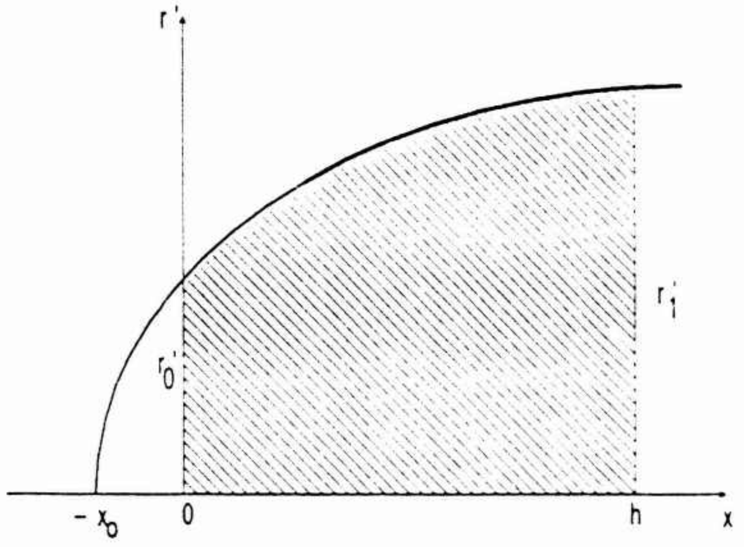


Fig.11: Volume of a Paraboloid Section.

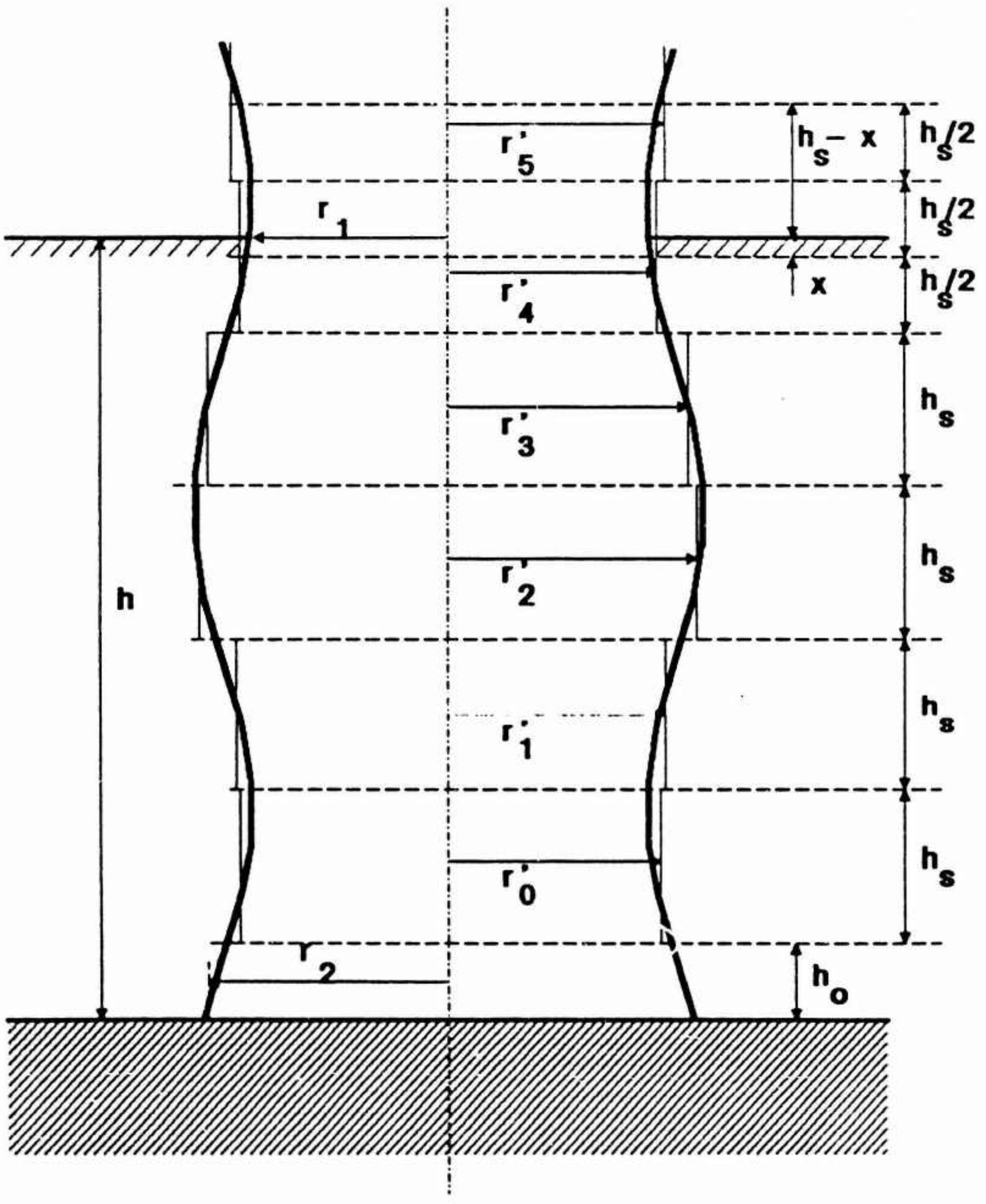


Fig. 12: Interpolation Algorithm for the Evaluation of the Crystal Diameter at the Boric Oxide Encapsulant Surface and of the Volume Immersed.

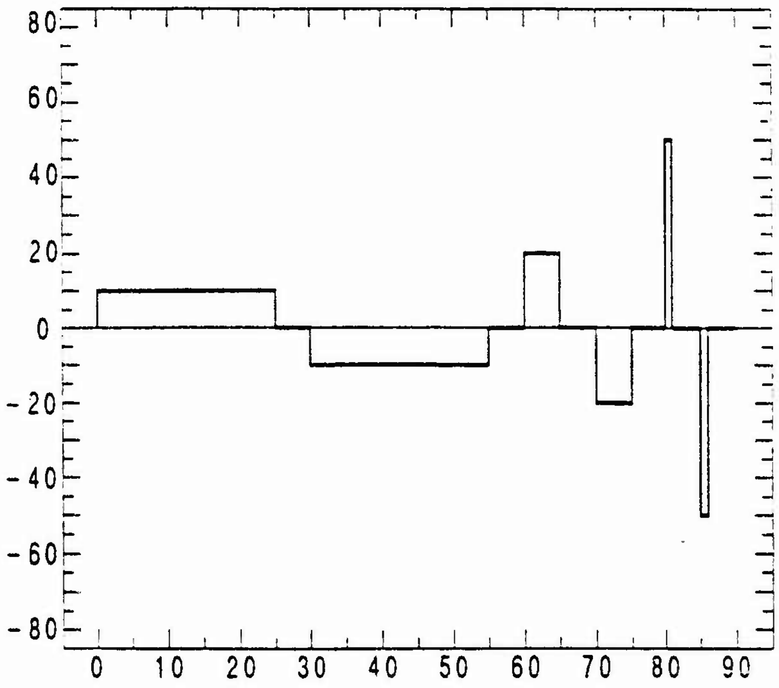


Fig. A1: "Actual" input signal used for the simulations.

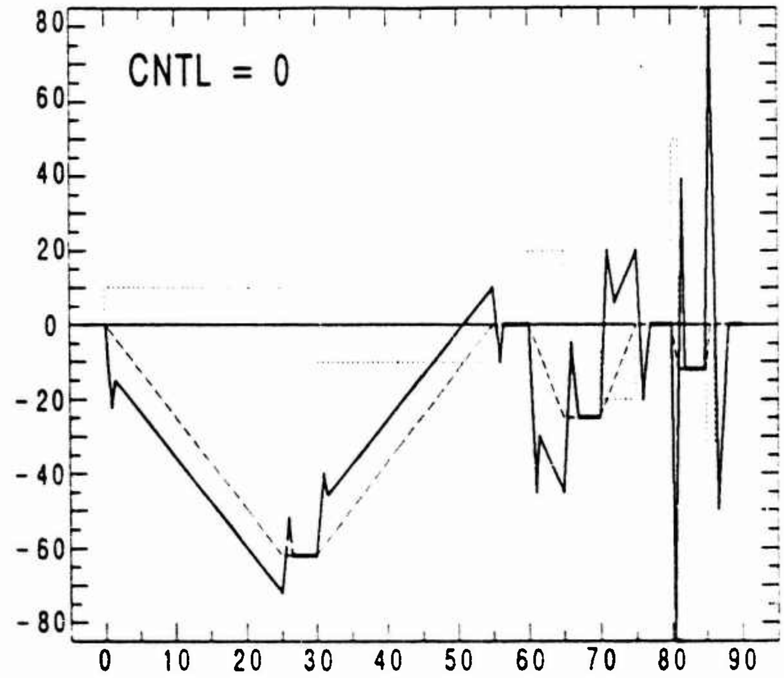


Fig. A2: Controller output signal (full line) and error integral (broken line) for unlimited operation with no option active.

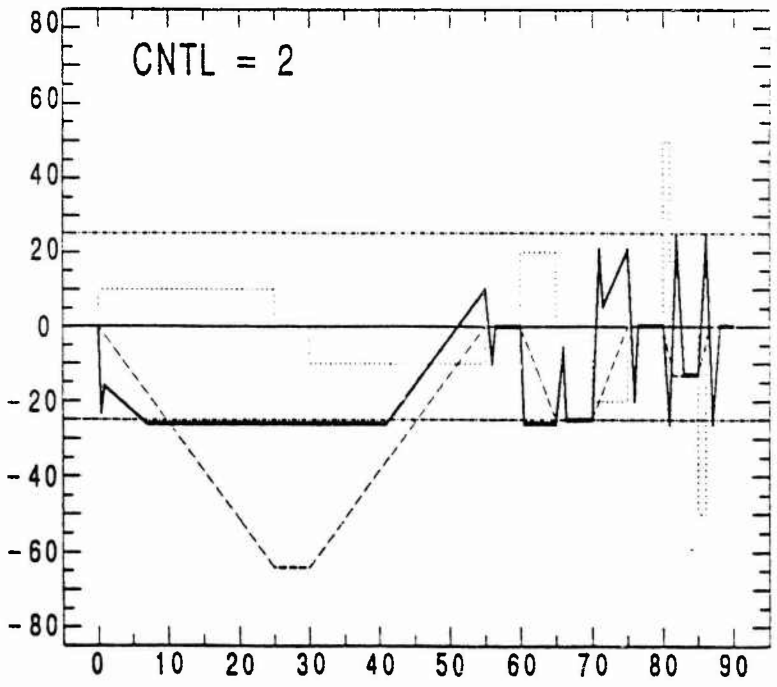


Fig. A3: Controller output signal (full line) and error integral (broken line) for output signal limiting with no anti-windup.

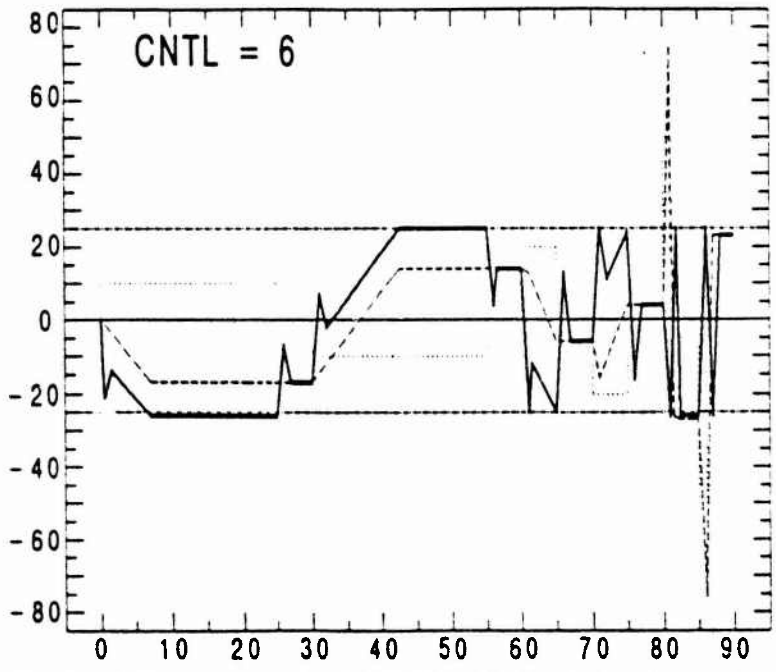


Fig. A4: Controller output signal (full line) and error integral (broken line) for output signal limiting with anti-windup Mode A.

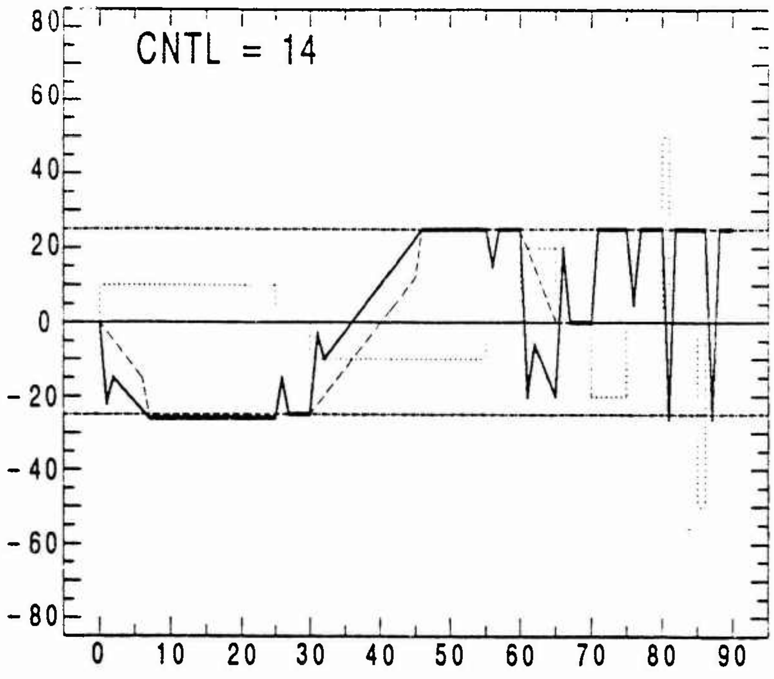


Fig. A5: Controller output signal (full line) and error integral (broken line) for output signal limiting with anti-windup Mode B.

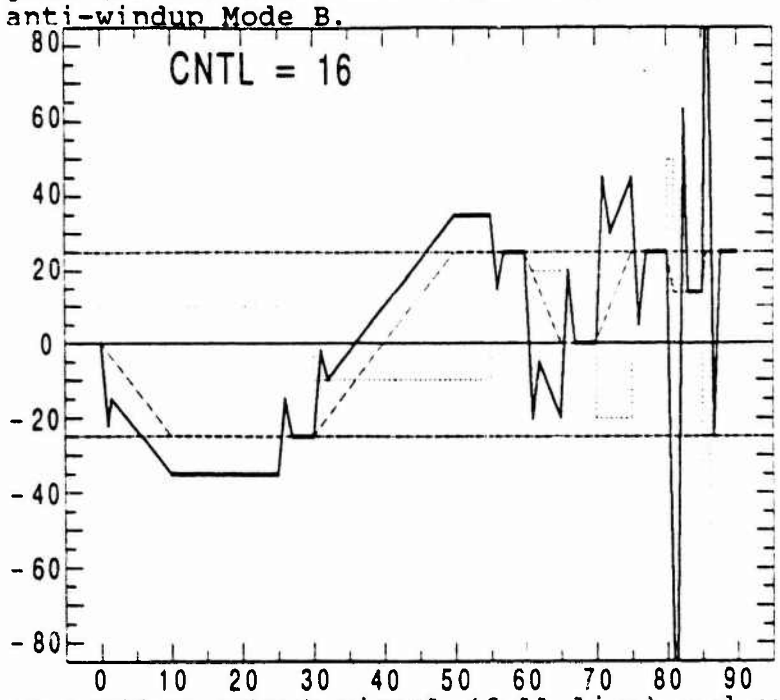


Fig. A6: Controller output signal (full line) and error integral (broken line) for integral limiting but no output signal limiting.

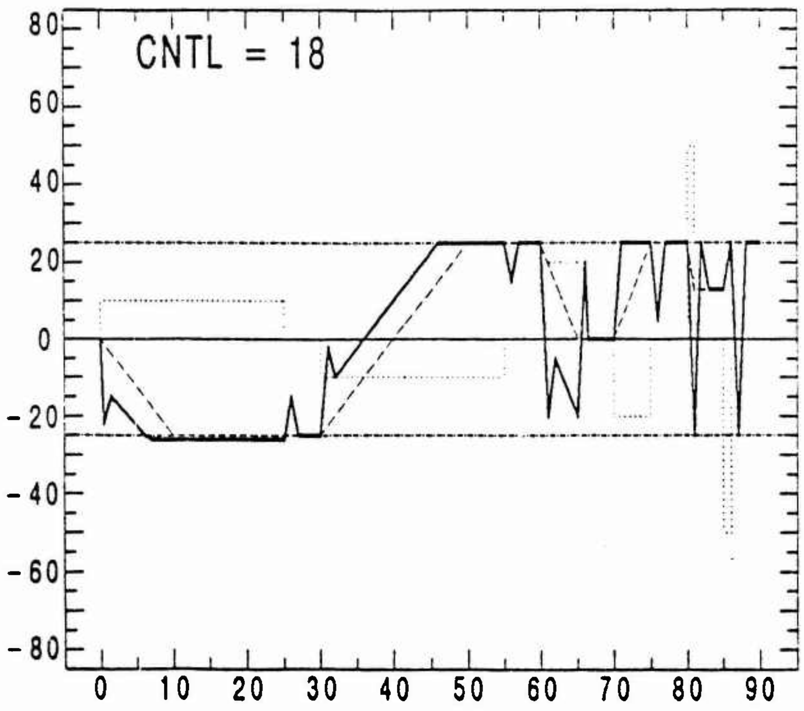


Fig. A7: Controller output signal (full line) and error integral (broken line) for integral and output signal limiting.